



**US Army Corps
of Engineers®**

Engineer Research and
Development Center

Building Composer

The Development of an Object Model for Facility Planning and Design Based on Customer Criteria

Jeffery S. Heckel

November 2002



REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188		
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YYYY) 01-11-2002		2. REPORT TYPE Final		3. DATES COVERED (FROM - TO) xx-06-2002 to xx-11-2002	
4. TITLE AND SUBTITLE Building Composer: The Development of an Object Model for Facility Planning and Design Based on Customer Criteria Unclassified			5a. CONTRACT NUMBER		
			5b. GRANT NUMBER		
			5c. PROGRAM ELEMENT NUMBER		
6. AUTHOR(S) Heckel, Jeffrey S. ; Author Wolfe, William J. ; Editor			5d. PROJECT NUMBER 622784AT41		
			5e. TASK NUMBER		
			5f. WORK UNIT NUMBER		
7. PERFORMING ORGANIZATION NAME AND ADDRESS U.S. Army Engineer Research and Development Center (ERDC) Construction Engineering Research Laboratory (CERL) PO Box 9005 Champaign, IL61826-9005			8. PERFORMING ORGANIZATION REPORT NUMBER ERDC/CERL TR-02-28		
9. SPONSORING/MONITORING AGENCY NAME AND ADDRESS Headquarters, U.S. Army Corps of Engineers (HQUSACE) 441 G St., NW. Washington, DC20314-1000			10. SPONSOR/MONITOR'S ACRONYM(S) CECW-EE		
			11. SPONSOR/MONITOR'S REPORT NUMBER(S)		
12. DISTRIBUTION/AVAILABILITY STATEMENT APUBLIC RELEASE					
13. SUPPLEMENTARY NOTES Copies are available from the National Technical Information Service, 5285 Port Royal Road, Springfield, VA 22161.					
14. ABSTRACT Building Composer is an application for use by planners, designers, and engineers during all phases of a facility's life cycle to create: (1) a project brief (architectural program), (2) a preliminary floor plan, and (3) a selection of systems and components. Building Composer allows customer-specific criteria to be specified at any appropriate level of detail. Various design wizards and analysis wizards perform specific actions based on these criteria to produce an Integrated Model-Based Facility Design. Criteria Composer refers to the portion of Building Composer used to create a project brief. It includes information regarding the total area of the project and some allocation of area to specific architectural functions, such as circulation and offices. The level of detail may vary from project to project; however, it contains enough information to generate a preliminary cost estimate. This work developed an object model to store information related to the planning and design of a facility, to be used as the basis for the development of Building Composer and Criteria Composer, to input, display, and manipulate this data.					
15. SUBJECT TERMS Building Composer; facility planning; design criteria; facility design; modeling					
16. SECURITY CLASSIFICATION OF:		17. LIMITATION OF ABSTRACT Same as Report (SAR)	18. NUMBER OF PAGES 130	19. NAME OF RESPONSIBLE PERSON Wolfe, William William.J.Wolfe@erdc.usace.army.mil	
a. REPORT Unclassified	b. ABSTRACT Unclassified	c. THIS PAGE Unclassified		19b. TELEPHONE NUMBER International Area Code Area Code Telephone Number 217352-6511 DSN -	
				Standard Form 298 (Rev. 8-98) Prescribed by ANSI Std Z39.18	

Foreword

This study was conducted for Headquarters, U.S. Army Corps of Engineers (HQUSACE) under Project 4A162784AT41, "Military Facilities Engineering Technology"; Work Unit A00, "Advanced Collaboration Repositories for Engineering Processes." The technical monitor was Jean McGinn, CECW-EE.

The work was performed by the Engineering Processes Branch (CF-N), of the Facilities Division (CF), Construction Engineering Research Laboratory (CERL). The CERL Principal Investigator was Jeffery S. Heckel. Recognition is due to Wayne Smith, David Stigberg, Van Woods, and William Zwicky for their assistance in testing and making suggestions to the code development of this research. Recognition is also given to Beth Brucker, Eric Griffith, and Kirk McGraw for their project planning and suggestions to the final Criteria Composer application. The technical editor was William J. Wolfe, Information Technology Laboratory. Dr. Michael P. Case is Chief, CECER-CF-N; L. Michael Golish is Operations Chief, CECER-CF. The associated Technical Director was Paul A. Howdyshell, CEERD-CV-ZT. The Acting Director of CERL is Dr. Alan W. Moore.

CERL is an element of the U.S. Army Engineer Research and Development Center (ERDC), U.S. Army Corps of Engineers. The Director of ERDC is Dr. James R. Houston and the Deputy to the Commander is A.J. Roberto, Jr.

DISCLAIMER

The contents of this report are not to be used for advertising, publication, or promotional purposes. Citation of trade names does not constitute an official endorsement or approval of the use of such commercial products. All product names and trademarks cited are the property of their respective owners.

The findings of this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

DESTROY THIS REPORT WHEN IT IS NO LONGER NEEDED. DO NOT RETURN IT TO THE ORIGINATOR.

Contents

Foreword	2
List of Figures	5
1 Introduction	7
Background	7
Objective	7
Approach	8
Mode of Tech Transfer	8
2 Overall Description	9
3 Object Model	10
Run EXPRESS file through StepTools application	10
Modify Generated Classes	11
Modify Class and Interface Hierarchy	13
<i>MDSAppInst</i>	17
<i>Other Custom Classes</i>	21
Implement Inverse Relationship Attributes	28
Add New Classes	31
<i>MDSFunction, MDSFunctionTemplate, and MDSFunctionInstance</i>	31
<i>MDSProgram</i>	32
<i>MDSProperty, MDSRequirement, and MDSAttribute</i>	34
<i>MDSRequirementSet</i>	35
Database Interface	35
Controller and Helper Classes	37
<i>MDSModelController</i>	38
<i>MDSProjectController</i>	38
<i>MDSObjectLinker</i>	38
PSE Post-processing	39
Object Model Use	39
4 Building Composer Core Library	43
Common Graphical Components and Utilities	43
<i>Library Classes</i>	43
<i>DataNode Class</i>	44

Project Tree.....	45
Property Panels.....	48
Requirements.....	52
Requirement Grid.....	52
Extensions.....	56
Common Commands	57
5 Criteria Composer Application.....	61
6 Future Work.....	62
References	64
Appendix A: IFC 1.50 EXPRESS File	65
Appendix B: Modified Generated Files	108
Appendix C: Custom Class Code	109
Appendix D: PSE Post-Processing Batch File.....	122
Appendix E: Microsoft Access Tables.....	123
Appendix F: Additional Information on Library Creation and Usage	126
CERL Distribution	128
Report Documentation Page	129

List of Figures

Figures

1	Criteria Composer core library graphical pieces	9
2	Class hierarchy before in the EXPRESS file	13
3	Result of Step Tools process	13
4	The three inverse attributes: "Contains," "IsContainedBy," and "ServicedBySystems"	29
5	How a site, building, and story are related through instances of the IfcRelContains class	29
6	Relation between MDSFunction, MDSFunctionTemplate, and MDSFunctionInstance	31
7	Site and story objects shown with their program objects	33
8	Relationship between MDSPROPERTY, MDSRequirement, and MDSAttribute classes	34
9	Property panel classes	48
10	Requirement set	53

1 Introduction

Background

Building Composer is an application for use by planners, designers, and engineers during all phases of a facility's life cycle. Its capabilities may be broadly categorized into three areas: creation of a project brief (architectural program), creation of a preliminary floor plan, and selection of systems and components. The core concept of *Building Composer* is that customer-specific criteria are specified at the appropriate level of detail, from the project to the individual space. For example, a customer should be able to specify that district steam heat be used on a project, that masonry load bearing walls be used on a building, that 32Watt T-8 florescent lights be used in corridors, or that a particular room be painted Oyster White. Various design wizards and analysis wizards perform specific actions based upon these criteria to produce an Integrated Model-Based Facility Design.

Criteria Composer is the name used to describe the portion of *Building Composer* used to create a project brief. It includes information regarding the total area of the project and some allocation of area to specific architectural functions, such as circulation and offices. The level of detail varies from project to project; however, it contains enough information to generate a preliminary cost estimate. It is acceptable to create a project that contains a list of architectural functions and their allocated areas without deciding how many buildings will be required. On the other hand, the planner may create a project with detailed information, such as the number of buildings and the number of stories in each building. Typically, the planner will not create such a detailed brief from scratch, but will copy it from a library and modify it as required to suit the customer's needs.

Objective

The specific objective of this work was to develop an object model to store information related to the planning and design of a facility, to be used as the basis for the development of an application, *Building Composer* and Criteria Composer, to input, display, and manipulate this data.

Approach

An object model that supports facility planning and design was developed. A core set of functionality was developed that can be used to create a final application that can be easily extended and customized by the application developer.

Mode of Tech Transfer

This technology will be transferred to Corps districts through the Totally Integrated Project Delivery (TIPD) initiative pilot program. To transfer the research behind criteria-based design into the industry design process, ERDC is participating in industry initiatives such as FIATEC, a joint Construction Industry Institute (CII) effort, Building Lifecycle Interoperable Software (BLIS) and the International Alliance for Interoperability (IAI). Customers (Department of State, Army Reserve, and National Guard) continue to support products in this area and are working on developing criteria-based design standards.

2 Overall Description

The Criteria Composer application is composed of three separate pieces: the underlying object model, the *Building Composer* core library, and the Criteria Composer files. The object model provides for the objects that are constructed and stored persistently within the database. On top of the object model is the *Building Composer* core library. This piece defines the software architectural picture of the main Criteria Composer application. This core library defines how other add-ons hook into and interact with the core library. The Criteria Composer files consist of those that create the add-on and define how to present the core library graphical pieces (Figure 1).

The following report discusses these three pieces and their important aspects. The audience for this report is not the casual user of either of these pieces, but is aimed at developers that would have to maintain, upgrade, or perform some other modifications to the underlying code base. There is no way that this report can review and discuss all of the details within these three pieces. It is hoped that this report will provide a good starting point for the developers and maintainers of this code base.

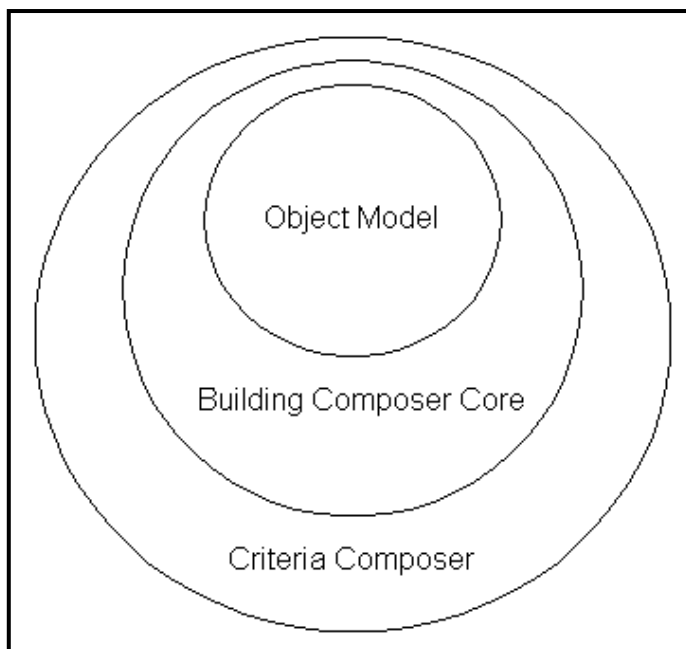


Figure 1. Criteria Composer core library graphical pieces.

3 Object Model

The underlying object model used is based on the Industry Foundation Classes (IFC) Version 1.5. These classes are defined by the International Alliance for Interoperability (IAI). The IAI is an action-oriented, not-for-profit organization. Its mission is to define, publish, and promote specifications for IFC as a basis for project information sharing in the building industry (architecture, engineering, construction, and facilities management). The information sharing is worldwide, throughout the project life cycle, and across all disciplines and technical applications. The IFC object model was chosen because of its completeness at the time of initial development. This model best represents the elements found in a facility design and can be extended by a developer. This, in addition to use of the development tools StepTools, made the IFC object model easier to work with. The use of StepTools would help jump-start the Java development by transforming EXPRESS entities to Java classes. The IFC object model file is written in the EXPRESS syntax. In addition, StepTools provides the code needed to import and export EXPRESS entities from a Part 21 file.

Several steps transform the raw IFC EXPRESS file into the Java classes that are used in the final object model:

1. Run the EXPRESS file through the StepTools application
2. Modify the generated classes
3. Modify the resulting class and interface hierarchy
4. Implement the inverse relationship attributes
5. Add new classes.

Run EXPRESS file through StepTools application

The first step will turn the EXPRESS file entities into a set of Java class files. The Standard Data Access Interface (SDAI) Java binding from StepTools takes an early-bound approach. Java classes are created for all of the entities and types in an EXPRESS schema to hold data. The EXPRESS to Java translator requires the EXPRESS parse data and ROSE meta-data for an EXPRESS schema, so these need to be generated first. The name of the IFC EXPRESS file

is IFC150Final.exp. (Appendix A give for a complete listing of the IFC EXPRESS file).

This will generate schema information in a file called <schema>.rose, in this case IFC150Final.rose.

The ST-Developer Advanced EXPRESS compiler expxfront is used to generate parse data used by the EXPRESS to Java compiler. This data is stored in the file IFC150Final_EXPX.rose. Now we can run the class generator.

Here the argument is the name of the parse data file, not the name of the EXPRESS file. This generates a Java class and interface files for the types and entities in the schema in subdirectories SDAI.<packages> and SDAI.COM.steptools.<packages>.

Modify Generated Classes

The automatically generated classes from StepTools are minimal. There is only a default no parameter constructor and no business logic created. Added to this is the complexity that is built into the IFC object model. Though a fairly complete model of a facility, it is also very flexible in its use. As a result of this flexibility, the model can be complex to use. In an effort to make the model easier to use, parameter constructors, additional debugging methods, and “helper” wrapper methods were added to some generated classes. Below are a few examples of this added code. (Appendix B includes a complete listing of files modified).

Here is code added to the Clfccartesianpoint class for a more complete toString method and additional constructors:

```
public String toString() {
    String temp = new String("[ " + getGUID() + ".Ver." + getVersion() + ", {");
    if (Coordinates != null) {
        for (int i = 0 ; i < Coordinates.length; ++i) {
            temp += Coordinates[i];
            if (i != Coordinates.length-1) temp += ",";
        }
        temp += "}]";
        return temp;
    }
}

public Clfccartesianpoint(double x, double y, double z) {
    setCoordinates(new double[] {x, y, z});
}

public Clfccartesianpoint(double x, double y) {
    setCoordinates(new double[] {x, y});
}

public Clfccartesianpoint(double[] coords) {
    setCoordinates(coords);
}
```

Here is code added to the Clfcproductshape class to add methods to make construction of a bounding box and its manipulation easier:

```

/** Constructs a bounding box product shape.
 * The bounding box position is set to "new Clfcaxis2placement3d(new Clfccartesianpoint(0, 0, 0))". */
public static Clfcproductshape productshape_boundingBox_create(double xSize, double ySize, double zSize) {
    Clfcboundingbox box = new Clfcboundingbox(new Clfcaxis2placement3d(new Clfccartesianpoint(0, 0, 0)), xSize, ySize, zSize);
    Clfcshaperepresentation shapeRep = new Clfcshaperepresentation();
    shapeRep.setRepresentationtype(Clfcshaperepresentationenum.BOUNDINGBOX);
    shapeRep.setItems(new Clfcgeometricrepresentationitem[] {box});
    Clfcshapebody body = new Clfcshapebody();
    body.setRepresentations(new Clfcshaperepresentation[] {shapeRep});
    Clfcproductshape prodShape = new Clfcproductshape();
    prodShape.getProductcomponentshape().setClfcshapebody(body);
    return prodShape;
}

public boolean productshape_isA_boundingBox() {
    boolean result = false;
    if (getProductcomponentshape().isClfcshapebody() == SDAI.lang.LOGICAL.TRUE) {
        Clfcshapebody body = getProductcomponentshape().getClfcshapebody();
        Clfcshaperepresentation[] shapeReps = body.getRepresentations();
        if (shapeReps != null) {
            for (int i = 0; i < shapeReps.length; i++) {
                if (shapeReps[i] != null) {
                    if (shapeReps[i].getRepresentationtype() == Clfcshaperepresentationenum.BOUNDINGBOX) {
                        result = true;
                    }
                }
            }
        }
    }
    return result;
}

public Clfcboundingbox productshape_boundingBox_get() {
    Clfcboundingbox result = null;
    if (productshape_isA_boundingBox() == false) return null;
    if (getProductcomponentshape().isClfcshapebody() == SDAI.lang.LOGICAL.TRUE) {
        Clfcshapebody body = getProductcomponentshape().getClfcshapebody();
        Clfcshaperepresentation[] shapeReps = body.getRepresentations();
        if (shapeReps != null) {
            for (int i = 0; i < shapeReps.length; i++) {
                if (shapeReps[i] != null) {
                    if (shapeReps[i].getRepresentationtype() == Clfcshaperepresentationenum.BOUNDINGBOX) {
                        Clfcgeometricrepresentationitem[] geoReps = shapeReps[i].getItems();
                        result = (Clfcboundingbox)geoReps[0];
                    }
                }
            }
        }
    }
    return result;
}

public void productshape_boundingBox_setXdim(double v) {
    if (productshape_isA_boundingBox() == false) return;
    productshape_boundingBox_get().setXdim(v);
}

public void productshape_boundingBox_setYdim(double v) {
    if (productshape_isA_boundingBox() == false) return;
    productshape_boundingBox_get().setYdim(v);
}

public void productshape_boundingBox_setZdim(double v) {
    if (productshape_isA_boundingBox() == false) return;
    productshape_boundingBox_get().setZdim(v);
}

public double productshape_boundingBox_getXdim() {
    if (productshape_isA_boundingBox() == false) return -1;
    return productshape_boundingBox_get().getXdim();
}

```

```

}

public double productshape_boundingBox_getYdim() {
    if (productshape_isA_boundingBox() == false) return -1;
    return productshape_boundingBox_get().getYdim();
}

public double productshape_boundingBox_getZdim() {
    if (productshape_isA_boundingBox() == false) return -1;
    return productshape_boundingBox_get().getZdim();
}

```

Modify Class and Interface Hierarchy

One of the results of the StepTools process is that the resulting class hierarchy is “flattened” out. For example, if there were this hierarchy before in the EXPRESS file (Figure 2), the result would be the structure shown in Figure 3.

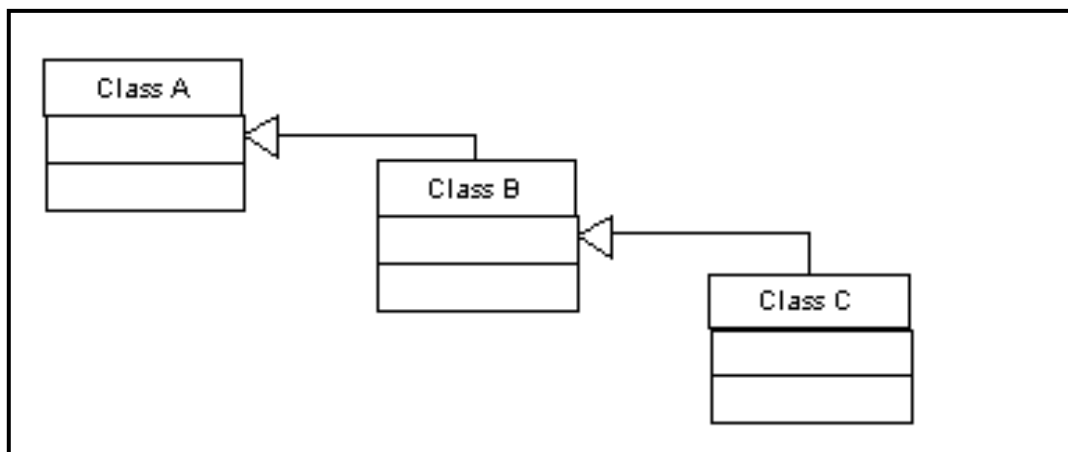


Figure 2. Class hierarchy before in the EXPRESS file.

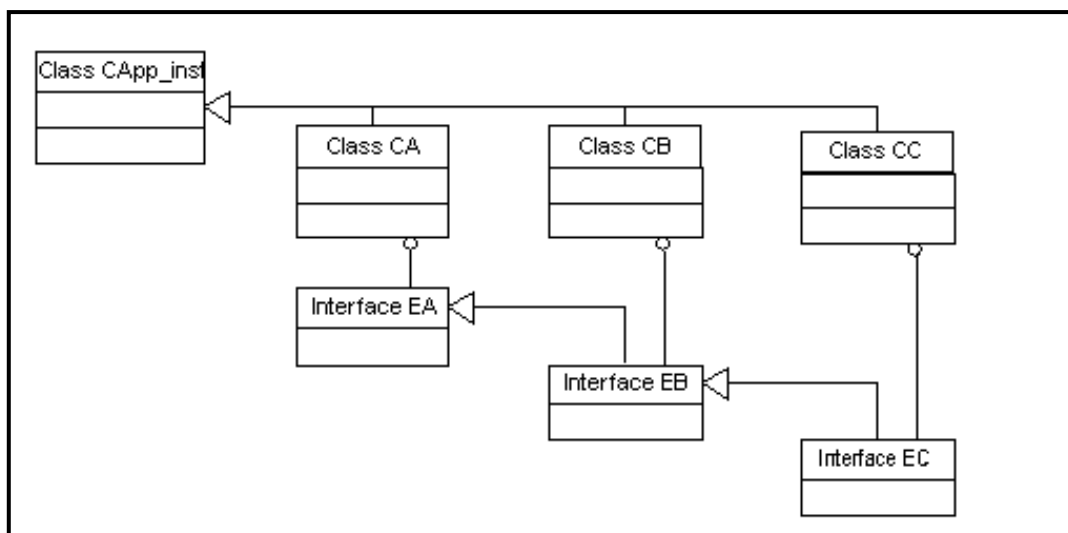


Figure 3. Result of Step Tools process.

Each class file generated (a “C” is added before the class name) has a corresponding interface file generated (an “E” is added before the class name) that defines the method signatures for the class. The code for these methods is then created in each class that implements an interface. In the example above, the code for any methods in Class A would be repeated in Class A, Class B, and Class C. This “flattening” of the object model results from the fact that an EXPRESS file can define classes with multiple inheritances. Java, on the other hand, cannot do multiple inheritances. Therefore StepTools flattens the actual Java class files, but maintains the inheritance through the Java interface files. Internally the Java classes pass and return interface objects from methods to allow this new hierarchy to work. For example:

```
public class Clfcsite implements Elfcsite {
    public final Elfcoccurrencepropertyset[] getOccurrenceproperties()
    public final void setOccurrenceproperties(Elfcoccurrencepropertyset[] v)
}
```

The “flattening” out of the object model makes modification of code and addition of new functionality difficult. It is necessary to modify the code in several places. To ease this problem, several custom classes are added within the hierarchy to try and restore the original hierarchy.

The following shows a more complex example of how some of the IFC object model is transformed through the StepTools process. The following shows the EXPRESS code for an IfcRoot, IfcObject, and IfcProject object:

```
ENTITY IfcRoot
ABSTRACT SUPERTYPE OF( ONEOF(
    ,IfcObject
    ,IfcRelationship
    ,IfcModelingAid));
    ProjectId      : IfcProjectUniqueld;
UNIQUE
    UR1: ProjectId;
END_ENTITY;
ENTITY IfcObject
ABSTRACT SUPERTYPE OF( ONEOF(
    ,IfcProduct
    ,IfcProcess
    ,IfcControl
    ,IfcDocument
    ,IfcGroup
    ,IfcProject
    ,IfcProxy
    ,IfcResource))
SUBTYPE OF (IfcRoot);
    OwnerHistory      : IfcOwnerHistory;
    TypeDefinitions   : LIST [0..?] OF IfcPropertyTypeDef;
    OccurrenceProperties : LIST [0..?] OF IfcOccurrencePropertySet;
    ExtendedProperties : LIST [0..?] OF IfcPropertySet;
INVERSE
    PartOfGroups      : SET [0..?] OF IfcRelGroups
WHERE
    WR1: SIZEOF(QUERY(temp <= ExtendedProperties |
        ('IFC150FINAL.IFCOCCURRENCEPROPERTYSET' IN TYPEOF(temp))
        OR
        ('IFC150FINAL.IFCSHAREDPROPERTYSET' IN TYPEOF(temp))))
        = 0;
END_ENTITY;
ENTITY IfcProject
```

```

SUBTYPE OF (IfcObject);
GlobalId      : OPTIONAL IfcGloballyUniqueId;
UnitsInContext : IfcUnitAssignment;
ProjectTeam   : IfcProjectTeamRegistry;
ProjectApps   : IfcProjectAppRegistry;
Classification : OPTIONAL IfcClassificationList;
AbsolutePlacement : IfcAxis2Placement;
END_ENTITY;

```

The following shows some of the resulting Java files generated (the class files for IfcRoot and IfcObject are not shown):

```

public interface Elfcroot extends Int_MDSAppInst {
    java.lang.String getProjectid();
    void setProjectid(java.lang.String v);
}

public interface Elfcobject extends Elfcroot {
    Elfcpropertytypedef[] getTypedefinitions();
    void setTypedefinitions(Elfcpropertytypedef[] v);
    Elfcoccurrencepropertyset[] getOccurrenceproperties();
    void setOccurrenceproperties(Elfcoccurrencepropertyset[] v);
    Elfcrlgroups[] getPartofgroups();
    Elfcpropertyset[] getExtendedproperties();
    void setExtendedproperties(Elfcpropertyset[] v);
    Elfcownerhistory getOwnerhistory();
    void setOwnerhistory(Elfcownerhistory v);
}

public interface Elfcproject extends Elfcobject {
    Elfcaxis2placement getAbsoluteplacement();
    void setAbsoluteplacement(Elfcaxis2placement v);
    java.lang.String getGlobalid();
    void setGlobalid(java.lang.String v);
    Elfcclassificationlist getClassification();
    void setClassification(Elfcclassificationlist v);
    Elfcprojectteamregistry getProjectteam();
    void setProjectteam(Elfcprojectteamregistry v);
    Elfcprojectappregistry getProjectapps();
    void setProjectapps(Elfcprojectappregistry v);
    Elfcunitassignment getUnitsincontext();
    void setUnitsincontext(Elfcunitassignment v);
}

public class Clfcproject implements Elfcproject {
    public Clfcproject() {
    }
    public Clfcproject(SDAI.lang.Model_contents mc) throws SDAI.lang.SdaiException {
        mc.moveInstance(this);
    }
    public String GetInstanceTypeBN() {
        return "Ifcproject";
    }
    public Class GetInstanceType() {
        return Elfcproject.class;
    }
    public EntityInfo getDomainInfo() {
        return domainInfo;
    }
    public static EntityInfo domainInfo = null;
    public final Elfcaxis2placement getAbsoluteplacement() {
        return Absoluteplacement;
    }
    public final void setAbsoluteplacement(Elfcaxis2placement v) {
        Absoluteplacement = v;
    }
    public final java.lang.String getGlobalid() {
        return Globalid;
    }
    public final void setGlobalid(java.lang.String v) {
        Globalid = v;
    }
    public final Elfcclassificationlist getClassification() {

```



```

        return(Elfcclassificationlist)Classification;
    }
    public final void setClassification(Elfcclassificationlist v) {
        Classification = v;
    }
    public final java.lang.String getProjectid() {
        return Projectid;
    }
    public final void setProjectid(java.lang.String v) {
        Projectid = v;
    }
    public final Elfcpropertytypedef[] getTypedefinitions() {
        return(Elfcpropertytypedef[])Typedefinitions;
    }
    public final void setTypedefinitions(Elfcpropertytypedef[] v) {
        Typedefinitions = v;
    }
    public final Elfcoccurrencepropertyset[] getOccurrenceproperties() {
        return(Elfcoccurrencepropertyset[])Occurrenceproperties;
    }
    public final void setOccurrenceproperties(Elfcoccurrencepropertyset[] v) {
        Occurrenceproperties = v;
    }
    public final Elfcrelgroups[] getPartofgroups() {
        throw new SdaiRuntimeException(SdaiException.FN_NAVL);
    }
    public final Elfcpropertyset[] getExtendedproperties() {
        return(Elfcpropertyset[])Extendedproperties;
    }
    public final void setExtendedproperties(Elfcpropertyset[] v) {
        Extendedproperties = v;
    }
    public final Elfcprojectteamregistry getProjectteam() {
        return(Elfcprojectteamregistry)Projectteam;
    }
    public final void setProjectteam(Elfcprojectteamregistry v) {
        Projectteam = v;
    }
    public final Elfcprojectappregistry getProjectapps() {
        return(Elfcprojectappregistry)Projectapps;
    }
    public final void setProjectapps(Elfcprojectappregistry v) {
        Projectapps = v;
    }
    public final Elfcunitassignment getUnitsincontext() {
        return(Elfcunitassignment)Unitsincontext;
    }
    public final void setUnitsincontext(Elfcunitassignment v) {
        Unitsincontext = v;
    }
    public final Elfcownerhistory getOwnerhistory() {
        return(Elfcownerhistory)Ownerhistory;
    }
    public final void setOwnerhistory(Elfcownerhistory v) {
        Ownerhistory = v;
    }
    public void copyInstance(App_inst a) {
        Elfcproject ca = (Elfcproject) a;
        Absoluteplacement = ca.getAbsoluteplacement();
        Globalid = ca.getGlobalid();
        Classification = ca.getClassification();
        Projectid = ca.getProjectid();
        Typedefinitions = (Elfcpropertytypedef[]) SDAUtil.makeCopy(ca.getTypedefinitions());
        Occurrenceproperties = (Elfcoccurrencepropertyset[]) SDAUtil.makeCopy(ca.getOccurrenceproperties());
        Extendedproperties = (Elfcpropertyset[]) SDAUtil.makeCopy(ca.getExtendedproperties());
        Projectteam = ca.getProjectteam();
        Projectapps = ca.getProjectapps();
        Unitsincontext = ca.getUnitsincontext();
        Ownerhistory = ca.getOwnerhistory();
    }
    public Elfcaxis2placement Absoluteplacement = new Clfcaxis2placement();
    public java.lang.String Globalid = null;
    public Object Classification = null;
    public java.lang.String Projectid = null;
    public Object[] Typedefinitions = null;
    public Object[] Occurrenceproperties = null;
    public Object[] Extendedproperties = null;

```

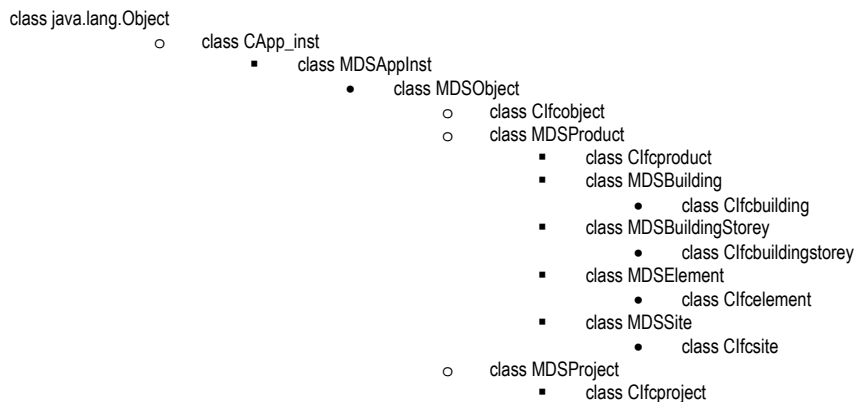
```

    public Object Projectteam = null;
    public Object Projectapps = null;
    public Object Unitsincontext = null;
    public Object Ownerhistory = null;
}

```

As can be seen, the interface files follow the hierarchy of the original IFC object model. The Clfcproject class file, on the other hand, does not extend from the Clfcobject class file—the class hierarchy has been “flattened.” The class files generated do have the code for their attribute accessor methods, but no business logic code is generated. Also note that the “INVERSE” attribute PartOfGroups in the IfcObject object did have a method generated in the Elfcobject interface, but the accessor method in the Clfcproject class would throw a runtime exception if called. It is left to the developer to complete the coding of these “INVERSE” attributes (see section “Implement Inverse Relationship Attributes” below).

To help make the addition of code easier and to add new needed functionality, new classes were created that are similar to the IFC classes, but added into the hierarchy tree. The code was placed in these new custom classes to help isolate them from the code in the automatically generated classes. This isolation would make later processes of the EXPRESS file through the StepTools application safer (the code may not be accidentally lost). These new classes were inserted into the class hierarchy before (as opposed to after) the IFC class so that an instance of the IFC object would have the added functionality. Here is a partial view of the IFC object model with the new custom classes inserted:



MDSAppInst

The first custom class created was MDSAppInst (interface Int_MDSAppInst). This class would become the new root class for all objects. To insert this class into the hierarchy, it is necessary to modify all classes that extend from CApp_inst to extend from MDSAppInst. It is also necessary to add the interface Int_MDSAppInst into the interface hierarchy. The modified class hierarchy is:

```

Class CApp_inst
> Class MDSAppInst extends CApp_inst implements Int_MDSAppInst
> Class Clfc2dcompositecurve extends MDSAppInst implements Elfc2dcompositecurve
> Class Clfcactorrole extends MDSAppInst implements Elfcactorrole
...

```

Addition of the MDSAppInst class allows for a central place to modify code that should be applied to all classes within the object model. There were several modifications that were done: automatic generation of a global unique identifier (GUID), ability to display debug information, and ability to export to an IFC file.

To accomplish the automatic generation of a GUID, a default public constructor was created. In this constructor, a GUID is generated from three parts. First is the name of the class, second is the count of objects created to this point during this running of the application, and third is a random number based on the time that this object is created. A sample GUID would be “Clfcproject_x0x_-3628410943273228335” for a Clfcproject object. Here is the code for the constructor and GUID generator is:

```

/** Default constructor. A m_guid is created during construction. */
public MDSAppInst() {
    setGUID(generateGUID(classNameFinalPortion()));
    count += 1; // update counter
}
/** Utility to return the final portion of a class name. If full class name is 'package1.package2.classA', this will return 'classA'. */
public final String classNameFinalPortion() {
    // get final portion of class name as base for m_guid
    return getClass().getName().substring(getClass().getName().lastIndexOf('.') + 1);
}
/** Routine to generate GUID. GUID is: base + "_x_" + count + "_x_" + random number */
private static String generateGUID(String base) {
    StringBuffer tmp = new StringBuffer(base);
    Random rand = new Random();
    tmp.append("_x_");
    tmp.append(count);
    tmp.append("_x_");
    tmp.append(rand.nextLong());
    return tmp.toString();
}

```

In addition, the IFC model has the concept of a GUID in a variable called the “Projectid.” Not every class has this variable. It was decided, however, to move this variable into the MDSAppInst class and apply it to all objects. It was therefore necessary to comment out this variable in several StepTools generated classes.

The next modification was the ability to display some simple debug information about an instance. The basic way of doing this was by overriding the “toString” method and implementing a new “dump” method:

```

public void dump() {
    System.out.print("== OBJECT == " + classNameFinalPortion() + ", " + toString() + "\n");
}
public String toString() {
    return new String("[ " + getGUID() + " :Ver:" + getVersion() + " ]");
}

```

A more advanced way of displaying debug information was also created. This used the Java reflection technique to examine an object at runtime and to determine the data to display. Using the reflection technique, it is not necessary to rewrite this code for every class:

```
// routine to print to system.out the internal field values
public void dumpInternals() {
    processClassInternals(true);
}
// routine to retrieve the internal field values as a vector of strings
public Vector getInternals() {
    return processClassInternals(false);
}

private Vector processClassInternals(boolean display) {
    Vector internals = new Vector();
    try {
        Class c = getClass();
        if (display == true) System.out.println("Instance " + this);
        if (display == true) System.out.println("Class " + c);
        processFieldInternals(internals, display, c, this);
        Class superC = c.getSuperclass();
        while (superC != null) {
            if (display == true) System.out.println("Superclass " + superC);
            if (superC == Class.forName("java.lang.Object"))
                superC = null;
            else if (superC == Class.forName("SDAI.COM.steptools.CApp_inst"))
                superC = null;
            else {
                processFieldInternals(internals, display, superC, this);
                superC = superC.getSuperclass();
            }
        }
    }
    catch (Exception exc) {
        exc.printStackTrace();
    }
    return internals;
}

private void processFieldInternals(Vector internals, boolean display, Class c, Object inst) {
    String field;
    try {
        Field[] fields = c.getDeclaredFields();
        for (int i = 0; i < fields.length; i++) {
            try {
                Object obj = fields[i].get(this);
                field = new String(c.getName() + "." + fields[i].getName() + ": " + obj);
                internals.addElement(field);
                if (display == true)
                    System.out.println(c.getName() + "." + fields[i].getName() + ": " + obj);
                if (obj != null) {
                    Class cl = obj.getClass();
                    if (obj.getClass().isArray()) {
                        if (display == true)
                            System.out.println("\t" + obj.getClass().getComponentType().getName() + "[" + Array.getLength(obj) + "]");
                    }
                }
            }
            catch (IllegalAccessException exc) {
                System.out.println(c + " IllegalAccessException, " + fields[i].getName());
            }
        }
    }
    catch (Exception exc) {
        exc.printStackTrace();
    }
}
```

The final modification was to add the capability to export the object (and its variables) to an IFC file. Through classes and methods inherited from StepTools, an object can be added to a repository that can then be written out to a file (see StepTools documentation for more details). The methods added employ the Java reflection technique, used in the advanced debug display described above, to make sure that all object variables are added to this repository. The following methods are added:

```
// routine to dump to a part 21 file this instance and any attribute object variables
public void dumpToIFC(SDAI.lang.Model_contents mc) {
    if (dumpingToIFC) return;
    dumpingToIFC = true;
    try {
        // move this object to the model contents
        if (this instanceof SDAI.lang.App_inst) {
            if (classNameFinalPortion().startsWith("Clfc") ||
                classNameFinalPortion().startsWith("Elfc")) {
                mc.moveInstance((SDAI.lang.App_inst)this);
            }
        }
        // go up heirarchy chain adding parent class fields
        Class superC = getClass().getSuperclass();
        while (superC != null) {
            if (superC == Class.forName("java.lang.Object"))
                superC = null;
            else if (superC == Class.forName("SDAI.COM.steptools.CApp_inst"))
                superC = null;
            else if (superC == Class.forName("SDAI.COM.steptools.Slfc150final.MDSAppInst"))
                superC = null;
            else {
                dumpToIFCHelper(superC, this, mc);
                superC = superC.getSuperclass();
            }
        }
        // add main class fields
        dumpToIFCHelper(getClass(), this, mc);
        dumpingToIFC = false;
    }
    catch (Exception exc) {
        System.out.println("Exception Dumping - 1: " + this);
        exc.printStackTrace();
        dumpingToIFC = false;
    }
}

private void dumpToIFCHelper(Class c, Object inst, Model_contents mc) {
    try {
        // check for any fields to be moved to the model contents
        Field[] fields = c.getDeclaredFields();
        for (int i = 0; i < fields.length; i++) {
            try {
                Object obj = fields[i].get(inst);
                if (obj != null) {
                    if (obj instanceof Int_MDSDumpToIFC) {
                        ((Int_MDSDumpToIFC)obj).dumpToIFC(mc);
                    }
                    if (obj.getClass().isArray()) {
                        for (int j = 0; j < Array.getLength(obj); j++) {
                            if (Array.get(obj, j) instanceof Int_MDSDumpToIFC) {
                                ((Int_MDSDumpToIFC)Array.get(obj, j)).dumpToIFC(mc);
                            }
                        }
                    }
                }
            }
            catch (IllegalAccessException exc) {
                System.out.println("Exception Dumping - 2: " + this);
                System.out.println("Illegal Access on Field: " + fields[i].getName() + "");
            }
        }
    }
}
```

```

        catch (Exception exc) {
            System.out.println("Exception Dumping - 3: " + this);
            exc.printStackTrace();
        }
    }
}

```

The dumpToIFC method had to be overridden in several classes to refine the exporting of an object and its variables to an IFC file.

Other Custom Classes

Next custom classes MDSObject (interface Int_MDSObject), MDSPProduct (interface Int_MDSPProduct), MDSProject (interface Int_MDSProject), MDSSite, MDSBuilding, MDSBuildingStorey, MDSSpace, MDS_Element, MDSRelationship1to1, and MDSRelationship1toN were created and added into the class and interface hierarchy.

MDSObject

The MDSObject class was created to add methods to support property capability to lower classes. This capability is provided through the use of the IFC IFCPropertySet objects and the “extendedProperties” attribute of the IFCObject object. The IFC objects and methods that provide property capability can be difficult to negotiate. The methods added in MDSObject provide a wrapper to this complexity to simplify the process of adding properties to an object. Here are a few of those methods:

```

/**
 * Returns true if the given descriptor exists in a default shared property set; MDSObject.MDSBasePropertySet.
 */
public boolean existsProperty(String descriptor) {
    if (isMDSBasePropertySet() == false) return false;
    Ifcpropertyset mdsBasePropSet = getMDSBasePropertySet();
    return getSimpleProperty(mdsBasePropSet, descriptor) != null;
}

/**
 * Removes the given descriptor from a default shared property set; MDSObject.MDSBasePropertySet.
 * This will remove all properties with the given descriptor.
 */
public void removeProperty(String descriptor) {
    if (isMDSBasePropertySet() == false) return;
    Ifcpropertyset mdsBasePropSet = getMDSBasePropertySet();
    Ifcsimpleproperty prop = getSimpleProperty(mdsBasePropSet, descriptor);
    if (prop != null) {
        removeSimpleProperty(mdsBasePropSet, prop);
    }
}

/**
 * Adds a new property in a default shared property set; MDSObject.MDSBasePropertySet.
 * No check is made first if the descriptor already exists!
 * Sets the value as an IFCDescriptiveMeasure.
 */
public void addStringProperty(String descriptor, String value) {
    if (isMDSBasePropertySet() == false) setupMDSBasePropertySet();
    Ifcpropertyset mdsBasePropSet = getMDSBasePropertySet();
    Ifcsimpleproperty prop = getSimpleProperty(mdsBasePropSet, descriptor);
    if (prop == null) {
        prop = new Ifcsimpleproperty();
    }
    prop.setDescriptor(descriptor);
}

```

```

        prop.getValueComponent().setIfcDescriptiveMeasure(value);
        addSimpleProperty(mdsBasePropSet, prop);
    }

/**
 * Adds a new property in a default shared property set; MDSObject.MDSBasePropertySet.
 * No check is made first if the descriptor already exists!
 * Sets the value as an IFCNumericMeasure.
 */
    public void addDoubleProperty(String descriptor, Double value) {
        if (isMDSBasePropertySet() == false) setupMDSBasePropertySet();
        IfcPropertySet mdsBasePropSet = getMDSBasePropertySet();
        IfcSimpleProperty prop = getSimpleProperty(mdsBasePropSet, descriptor);
        if (prop == null) {
            prop = new IfcSimpleProperty();
        }
        prop.setDescriptor(descriptor);
        prop.getValueComponent().setIfcNumericMeasure(value.doubleValue());
        addSimpleProperty(mdsBasePropSet, prop);
    }

/**
 * Gets a property in a default shared property set; MDSObject.MDSBasePropertySet.
 * Returns the first matching descriptor!
 * The return value is null if property not found.
 */
    public String getStringProperty(String descriptor) {
        if (isMDSBasePropertySet() == false) return null;
        IfcPropertySet mdsBasePropSet = getMDSBasePropertySet();
        IfcSimpleProperty prop = getSimpleProperty(mdsBasePropSet, descriptor);
        if (prop != null) {
            return prop.getValueComponent().getIfcDescriptiveMeasure();
        }
        return null;
    }

/**
 * Gets a property in a default shared property set; MDSObject.MDSBasePropertySet.
 * Returns the first matching descriptor!
 * The return value is null if property not found.
 */
    public Double getDoubleProperty(String descriptor) {
        if (isMDSBasePropertySet() == false) return null;
        IfcPropertySet mdsBasePropSet = getMDSBasePropertySet();
        IfcSimpleProperty prop = getSimpleProperty(mdsBasePropSet, descriptor);
        if (prop != null) {
            return new Double(prop.getValueComponent().getIfcNumericMeasure());
        }
        return null;
    }

/**
 * Gets a vector of descriptors in a default shared property set; MDSObject.MDSBasePropertySet.
 */
    public Vector getPropertyTitles() {
        Vector result = new Vector();
        if (isMDSBasePropertySet() == false) return result;
        IfcPropertySet mdsBasePropSet = getMDSBasePropertySet();
        IfcProperty[] props = mdsBasePropSet.getProperties();
        if (props != null) {
            for (int i = 0; i < props.length; i++) {
                result.addElement(((IfcSimpleProperty)props[i]).getDescriptor());
            }
        }
        return result;
    }
}

```

MDSProduct

The MDSProduct class was created to add methods to support geometric capability to lower classes. A default constructor creates a local placement location and a zero size bounding box for an object. As with IFC property support, there are geometry methods in the IFC object model. Again, however, these can be difficult to negotiate. MDSProduct defines methods to simplify some geometric ca-

pabilities for an object. The defined default constructor (and some other methods) are:

```
public abstract class MDSPProduct extends MDSObject implements Int_MDSPProduct {
/**
 * A local placement object is automatically created.
 * A bounding box product shape (size 0, 0, 0) is automatically created.
 */
    public MDSPProduct() {
        setLocalplacement(new Clfclocalplacement(new Clfcaxis2placement3d(new Clfccartesianpoint(0, 0, 0)));
        setProductshape(Clfcproductshape.productshape_boundingBox_create(0, 0, 0));
    }
//-----
// these methods are redefined in subclasses!
    public Elfclocalplacement getLocalplacement() {
        System.out.println("MDSPProduct:getLocalplacement, SHOULD NOT BE HERE!");
        return null;
    }
    public void setLocalplacement(Elfclocalplacement v) {
        System.out.println("MDSPProduct:setLocalplacement, SHOULD NOT BE HERE!");
        return;
    }
    public Elfcproductshape getProductshape() {
        System.out.println("MDSPProduct:getProductshape, SHOULD NOT BE HERE!");
        return null;
    }
    public void setProductshape(Elfcproductshape v) {
        System.out.println("MDSPProduct:setProductshape, SHOULD NOT BE HERE!");
        return;
    }
}
//-----
//=====
// LOCATION ROUTINES
//=====
    public void setRelativeLocation(Elfccartesianpoint location) {
        if (location == null) return;
        if (getLocalplacement() == null) return;
        ((Clfclocalplacement)getLocalplacement()).setRelativePlacementLocation(location);
    }
    public void setAbsoluteLocation(Elfccartesianpoint location) {
        if (location == null) return;
        if (getLocalplacement() == null) return;
        if (getLocalplacement().getPlacementreto() == null) return;
        if (getLocalplacement().getPlacementreto().isIfcproduct() == SDAI.lang.LOGICAL.TRUE)
            setRelativeLocation(Clfccartesianpoint.delete(location, getLocalplacement().getPlacementreto().getIfcproduct().getAbsoluteLocation()));
        setRelativeLocation(location);
    }
    public Elfccartesianpoint getRelativeLocation() {
        if (getLocalplacement() == null) return null;
        return((Clfclocalplacement)getLocalplacement()).getRelativePlacementLocation();
    }
    public Elfccartesianpoint getAbsoluteLocation() {
        if (getLocalplacement() == null) return null;
        if (getLocalplacement().getPlacementreto() == null) return null;
        if (getLocalplacement().getPlacementreto().isIfcproduct() == SDAI.lang.LOGICAL.TRUE)
            return Clfccartesianpoint.add(getLocalplacement().getPlacementreto().getIfcproduct().getAbsoluteLocation(), getRelativeLocation());
        return getRelativeLocation();
    }
}
//=====
// ANGLE ROUTINES
//=====
    public void setRelativeAngle(double angle) {
        if (getLocalplacement() == null) return;
        ((Clfclocalplacement)getLocalplacement()).setRelativePlacementAngle(angle);
    }
    public void setAbsoluteAngle(double angle) {
        if (getLocalplacement() == null) return;
        if (getLocalplacement().getPlacementreto() == null) return;
        if (getLocalplacement().getPlacementreto().isIfcproduct() == SDAI.lang.LOGICAL.TRUE)
            setRelativeAngle(angle - getLocalplacement().getPlacementreto().getIfcproduct().getAbsoluteAngle());
        setRelativeAngle(angle);
    }
    public double getRelativeAngle() {
        if (getLocalplacement() == null) return 0.0;
        return((Clfclocalplacement)getLocalplacement()).getRelativePlacementAngle();
    }
}
```



```

    public double getAbsoluteAngle() {
        if (getLocalplacement() == null) return 0.0;
        if (getLocalplacement().getPlacementrelto() == null) return 0.0;
        if (getLocalplacement().getPlacementrelto().isIfcproduct() == SDAI.lang.LOGICAL.TRUE)
            return (getLocalplacement().getPlacementrelto().getIfcproduct().getAbsoluteAngle() + getRelativeAngle());
        return getRelativeAngle();
    }
}

```

MDSProject, MDSSite, MDSBuilding, and MDSBuildingStorey

These classes were created to add methods to support accessors for higher and lower objects in the hierarchy, maintaining the object's program (see "MDSProgram," p 32), additional geometric support, inverse relationships (see "Implement Inverse Relationship Attributes," p 28), and the object's requirement sets (see "MDSRequirementSet," p 35). A typical example of the MDSProject class is:

```

public abstract class MDSProject
    extends MDSObject
    implements Int_InvContains, Int_MDSProgramContainer, Int_MDSProject, Int_MDSRequirementSetContainer {
//-----
// these methods are redefined in subclasses!
    public Elfcaxis2placement getAbsoluteplacement() {
        System.out.println("MDSProject:getAbsoluteplacement, SHOULD NOT BE HERE!");
        return null;
    }
//-----
    MDSProgram m_program;
    public MDSProject() {
        m_program = new MDSProgram(this);
    }
// =====
    /** Returns a Clfcsite object. */
    public Clfcsite getSite() {
        Vector sites = getContainees(Clfcsite.class);
        if (sites.size() > 0) {
            return (Clfcsite)sites.elementAt(0);
        }
        return null;
    }
// =====
    public void dump() {
        super.dump();
        if (m_contains != null) m_contains.dump();
        if (m_reqSetContainer != null) m_reqSetContainer.dump();
        System.out.print("\tLocation:");
        if (getRelativeLocation() != null) {
            System.out.print(" Relative: " + getRelativeLocation());
        }
        if (getAbsoluteLocation() != null) {
            System.out.print(" Absolute: " + getAbsoluteLocation());
        }
        System.out.print("\n");
        System.out.print("\tAngle:");
        System.out.print(" Relative: " + Double.toString(getRelativeAngle()));
        System.out.print(" Absolute: " + Double.toString(getAbsoluteAngle()));
        System.out.print("\n");
    }
//=====
// dumpToIFC Routine
//=====
    transient private boolean dumpingToIFC = false;
    // routine to dump to a part 21 file this instance and any attribute object variables
    public void dumpToIFC(SDAI.lang.Model_contents mc) {
        if (dumpingToIFC) return;
        dumpingToIFC = true;

        // allow MDSRequirementSetContainer to build property sets for requirement sets - NEEDS TO BE BEFORE CALL TO SUPER!
        m_reqSetContainer.exportToIFC();
        // dump parents

```

```

        super.dumpToIFC(mc);
        if (MDSIFCController.getInstance().exportProgramInfo) {
            // dump program
            getProgram().exportToIFC(getGUID() + " Project Program").dumpToIFC(mc);
        }
        dumpingToIFC = false;
    }
}

//=====
// Int_MDSProgramContainer IMPLEMENTATION
//=====

//=====
public MDSProgram getProgram() {
    return m_program;
}
//=====
/** Returns null. */
public MDSProgram getParentProgram() {
    return null;
}
//=====
/** Returns a Vector of site and building programs. */
public Vector getChildPrograms() {
    Vector result = new Vector();
    if (getSite() == null) {
        return result;
    }
    result.addElement(getSite().getProgram());
    Enumeration buildings = getSite().getBuildings().elements();
    while (buildings.hasMoreElements()) {
        MDSBuilding building = (MDSBuilding)buildings.nextElement();
        result.addElement(building.getProgram());
    }
    return result;
}
//=====
public double getPlannedArea(MDSFunctionTemplate template) {
    return getProgram().getPlannedArea(template);
}
}
public double getTotalPlannedArea(MDSFunctionTemplate template) {
    double result = getPlannedArea(template);
    if (getSite() == null) {
        return result;
    }
    result += getSite().getTotalPlannedArea(template);
    Enumeration buildings = getSite().getBuildings().elements();
    while (buildings.hasMoreElements()) {
        MDSBuilding building = (MDSBuilding)buildings.nextElement();
        result += building.getPlannedArea(template); // only get planned (not total)
        //result += building.getTotalPlannedArea(template);
    }
    return result;
}
//=====
public double getPlacedArea() {
    return getProgram().getPlacedArea();
}
}
public double getTotalPlacedArea() {
    double result = getPlacedArea();
    if (getSite() == null) {
        return result;
    }
    result += getSite().getTotalPlacedArea();
    Enumeration buildings = getSite().getBuildings().elements();
    while (buildings.hasMoreElements()) {
        MDSBuilding building = (MDSBuilding)buildings.nextElement();
        result += building.getTotalPlacedArea();
    }
    return result;
}
}
public double getPlacedArea(MDSFunctionTemplate template) {
    return getProgram().getPlacedArea(template);
}
}
public double getTotalPlacedArea(MDSFunctionTemplate template) {
    double result = getPlacedArea(template);
    if (getSite() == null) {
        return result;
    }
}

```

```

        result += getSite().getTotalPlacedArea(template);
        Enumeration buildings = getSite().getBuildings().elements();
        while (buildings.hasMoreElements()) {
            MDSBuilding building = (MDSBuilding)buildings.nextElement();
            result += building.getTotalPlacedArea(template);
        }
        return result;
    }

    public double getPlacedArea(MDSFunctionInstance instance) {
        return getProgram().getPlacedArea(instance);
    }

    public double getTotalPlacedArea(MDSFunctionInstance instance) {
        double result = getPlacedArea(instance);
        if (getSite() == null) {
            return result;
        }
        result += getSite().getTotalPlacedArea(instance);
        Enumeration buildings = getSite().getBuildings().elements();
        while (buildings.hasMoreElements()) {
            MDSBuilding building = (MDSBuilding)buildings.nextElement();
            result += building.getTotalPlacedArea(instance);
        }
        return result;
    }

    // =====
    public Vector getFunctionInstances(MDSFunctionTemplate template) {
        return getProgram().getFunctionInstances(template);
    }

    public Vector getTotalFunctionInstances(MDSFunctionTemplate template){
        Vector result = getFunctionInstances(template);
        if (getSite() == null) {
            return result;
        }
        Enumeration instances = getSite().getTotalFunctionInstances(template).elements();
        while (instances.hasMoreElements()) {
            result.addElement(instances.nextElement());
        }
        Enumeration buildings = getSite().getBuildings().elements();
        while (buildings.hasMoreElements()) {
            MDSBuilding building = (MDSBuilding)buildings.nextElement();
            instances = building.getTotalFunctionInstances(template).elements();
            while (instances.hasMoreElements()) {
                result.addElement(instances.nextElement());
            }
        }
        return result;
    }

    // =====
    public Vector getSpaces(MDSFunctionTemplate template) {
        return getProgram().getSpaces(template);
    }

    public Vector getTotalSpaces(MDSFunctionTemplate template) {
        Vector result = getSpaces(template);
        if (getSite() == null) {
            return result;
        }
        Enumeration spaces = getSite().getTotalSpaces(template).elements();
        while (spaces.hasMoreElements()) {
            result.addElement(spaces.nextElement());
        }
        Enumeration buildings = getSite().getBuildings().elements();
        while (buildings.hasMoreElements()) {
            MDSBuilding building = (MDSBuilding)buildings.nextElement();
            spaces = building.getTotalSpaces(template).elements();
            while (spaces.hasMoreElements()) {
                result.addElement(spaces.nextElement());
            }
        }
        return result;
    }

    public Vector getSpaces(MDSFunctionInstance instance) {
        return getProgram().getSpaces(instance);
    }

    public Vector getTotalSpaces(MDSFunctionInstance instance) {
        Vector result = getSpaces(instance);
        if (getSite() == null) {
            return result;
        }

```

```

    }
    Enumeration spaces = getSite().getTotalSpaces(instance).elements();
    while (spaces.hasMoreElements()) {
        result.addElement(spaces.nextElement());
    }
    Enumeration buildings = getSite().getBuildings().elements();
    while (buildings.hasMoreElements()) {
        MDSBuilding building = (MDSBuilding)buildings.nextElement();
        spaces = building.getTotalSpaces(instance).elements();
        while (spaces.hasMoreElements()) {
            result.addElement(spaces.nextElement());
        }
    }
    return result;
}

//=====
// Int_InvContains IMPLEMENTATION
//=====

    InvContains m_contains = new InvContains();
    public final Elfcrelcontains[] getContains() {
        return m_contains.getContains();
    }
    public final void addContains(Elfcrelcontains v) {
        m_contains.addContains(v);
    }
    public final void removeContains(Elfcrelcontains v) {
        m_contains.removeContains(v);
    }
    public final Elfcrelcontains linkContaineer(Int_InvIsContainedBy containeeObj) {
        Elfcrelcontains link = MDSControllers.MDSObjectLinker.link_containerElementWithContaineerElement-((Elfcrelcontains)this, (Elfcrelcontains)containeeObj);
        this.addContains(link);
        containeeObj.addIsContainedby(link);
        return link;
    }
    public final Vector unlinkContaineer(Int_InvIsContainedBy containeeObj) {
        return m_contains.unlinkContaineer(containeeObj);
    }
    public final Vector getContaineers() {
        return m_contains.getContaineers();
    }
    public final Vector getContaineers(Class containeeClass) {
        return m_contains.getContaineers(containeeClass);
    }
}

//=====
// Int_MDSPProject IMPLEMENTATION
//=====
//=====
// LOCATION ROUTINES
//=====

    public void setRelativeLocation(Elfccartesianpoint location) {
        if (getAbsoluteplacement() != null)
            ((Elfccaxis2placement)getAbsoluteplacement()).setAxisLocation(location);
    }
    public void setAbsoluteLocation(Elfccartesianpoint location) {
        setRelativeLocation(location);
    }
    public Elfccartesianpoint getRelativeLocation() {
        if (getAbsoluteplacement() != null)
            return((Elfccaxis2placement)getAbsoluteplacement()).getAxisLocation();
        return null;
    }
    public Elfccartesianpoint getAbsoluteLocation() {
        return getRelativeLocation();
    }
}

//=====
// ANGLE ROUTINES
//=====

    public void setRelativeAngle(double angle) {
        if (getAbsoluteplacement() != null)
            ((Elfccaxis2placement)getAbsoluteplacement()).setReferenceAngle(angle);
    }
    public void setAbsoluteAngle(double angle) {
        setRelativeAngle(angle);
    }
    public double getRelativeAngle() {
        if (getAbsoluteplacement() != null)
            return((Elfccaxis2placement)getAbsoluteplacement()).getReferenceAngle();
    }

```

```

        return 0.0;
    }
    public double getAbsoluteAngle() {
        return getRelativeAngle();
    }
}
//=====
// Int_MDSRequirementSetContainer IMPLEMENTATION
//=====
MDSRequirementSetContainer m_reqSetContainer = new MDSRequirementSetContainer(this);
public void addRequirementSet(MDSRequirementSet requirementSet) {
    m_reqSetContainer.addRequirementSet(requirementSet);
}
public boolean removeRequirementSet(MDSRequirementSet requirementSet) {
    return m_reqSetContainer.removeRequirementSet(requirementSet);
}
public void removeAllRequirementSets() {
    m_reqSetContainer.removeAllRequirementSets();
}
public Vector getRequirementSets() {
    return m_reqSetContainer.getRequirementSets();
}
}
public MDSRequirementSet findRequirementSet(String requirementSetName) {
    return m_reqSetContainer.findRequirementSet(requirementSetName);
}
}
}

```

Implement Inverse Relationship Attributes

Several class attributes defined in the EXPRESS file are for “Inverse” relationships. For example, object IfcBuilding is defined in EXPRESS as:

```

ENTITY IfcBuilding
SUBTYPE OF (IfcProduct);
    GenericType : IfcBuildingTypeEnum;
    calcTotalHeight : OPTIONAL IfcLengthMeasure;
    calcSiteCoverage : OPTIONAL IfcAreaMeasure;
    calcTotalVolume : OPTIONAL IfcVolumeMeasure;
INVERSE
    Contains : SET [0:?] OF IfcRelContains FOR RelatingObject;
    IsContainedBy : SET [0:?] OF IfcRelContains FOR RelatedObjects;
    ServicedBySystems: SET [0:?] OF IfcRelServicesBuildings FOR RelatedObjects;
END_ENTITY;

```

The three inverse attributes—“Contains,” “IsContainedBy,” and “ServicedBySystems”—are not implemented in the new object model. Inverse attributes imply a reference from one object to another. In this case, there should be a reference from a IfcRelContains object to a IfcBuilding object using the RelatingObject attribute from IfcRelContains. This implied attribute should be named Contains. These are two other implied references named IsContainedBy and ServicedBySystems. **Figure 4** illustrates this concept. In this example, the implied references from the Building object and the Story object to the RelContains object are not generated.

To implement some of these implied references, three new custom classes were created: InvConnectedToFrom (interface Int_InvConnectedToFrom), InvContains (interface Int_InvContains), and InvContainedBy (interface Int_InvContainedBy). In addition to these classes, a helper class MDSObject-

Linker was created. This class provided methods used by the custom classes to help query the database and filter results from relationship objects. Instances of `InvConnectedToFrom`, `InvContains`, or `InvContainedBy` are created in appropriate classes to delegate the responsibility of maintaining these inverse relationships. Figure 5 shows this situation for a site, building, and story and how they are related through instances of the `Ifcrelcontains` class.

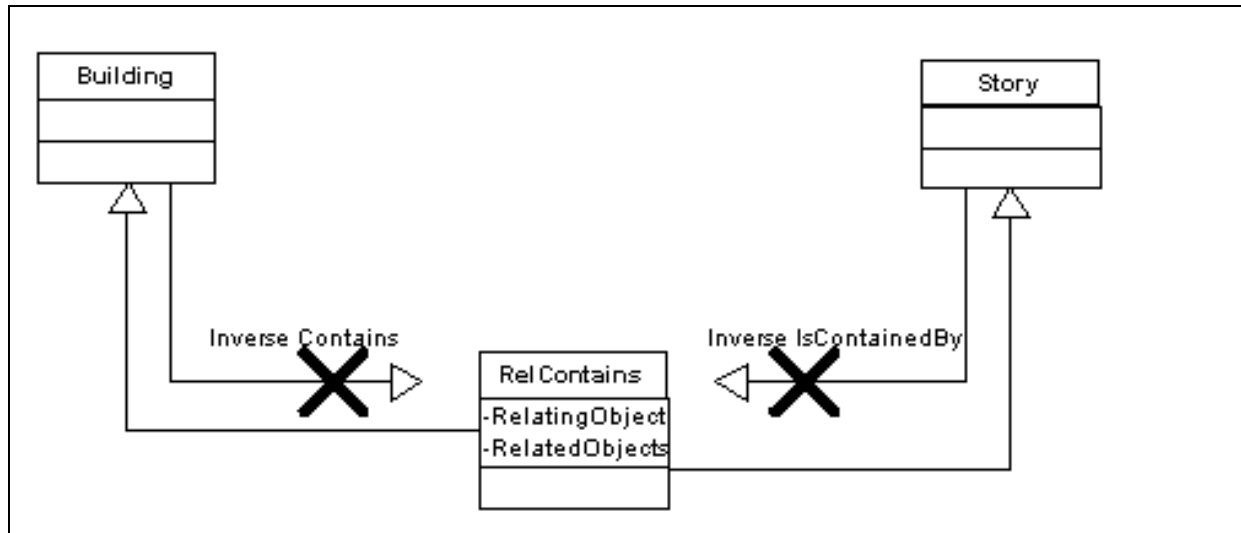


Figure 4. The three inverse attributes: “Contains,” “IsContainedBy,” and “ServedBySystems.”

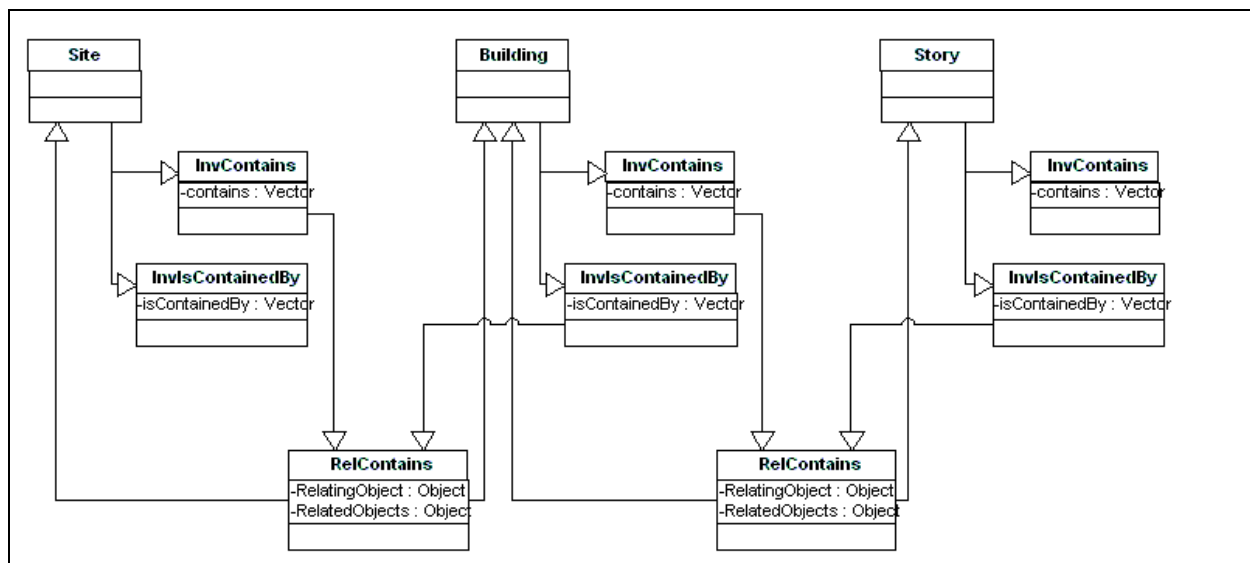


Figure 5. How a site, building, and story are related through instances of the `Ifcrelcontains` class.

The following code example shows how instances of `InvContains` and `InvContainedBy` are created and delegated to in `MDSBuilding` (the parent of `CIfcbuilding`):

```
public abstract class MDSBuilding
    extends MDSPProduct
    implements Int_InvContains, Int_InvIsContainedBy {
    ...
    //=====
    // Int_InvContains IMPLEMENTATION
    //=====
    InvContains m_contains = new InvContains();
    public final ElfcRelContains[] getContains() {
        return m_contains.getContains();
    }
    public final void addContains(ElfcRelContains v) {
        m_contains.addContains(v);
    }
    public final void removeContains(ElfcRelContains v) {
        m_contains.removeContains(v);
    }
    public final CIfcRelContains linkContaineer(Int_InvIsContainedBy containeeObj) {
        CIfcRelContains link =
MDSObjectLinker.link_containerElementWithContaineerElement((ElfcObject)this, (ElfcObject)containeeObj);
        this.addContains(link);
        containeeObj.addIsContainedby(link);
        return link;
    }
    public final Vector unlinkContaineer(Int_InvIsContainedBy containeeObj) {
        return m_contains.unlinkContaineer(containeeObj);
    }
    public final Vector getContaineers() {
        return m_contains.getContaineers();
    }
    public final Vector getContaineers(Class containeeClass) {
        return m_contains.getContaineers(containeeClass);
    }
    //=====
    // Int_InvIsContainedBy IMPLEMENTATION
    //=====
    InvIsContainedBy m_isContainedBy = new InvIsContainedBy();
    public final ElfcRelContains[] getIsContainedby() {
        return m_isContainedBy.getIsContainedby();
    }
    public final void addIsContainedby(ElfcRelContains v) {
        m_isContainedBy.addIsContainedby(v);
    }
    public final void removeIsContainedby(ElfcRelContains v) {
        m_isContainedBy.removeIsContainedby(v);
    }
    public final CIfcRelContains linkContainer(Int_InvContains containerObj) {
        CIfcRelContains link =
MDSObjectLinker.link_containerElementWithContaineerElement((ElfcObject)containerObj, (ElfcObject)this);
        containerObj.addContains(link);
        this.addIsContainedby(link);
        return link;
    }
    public final Vector unlinkContainer(Int_InvContains containerObj) {
        return m_isContainedBy.unlinkContainer(containerObj);
    }
    public final Vector getContainers() {
        return m_isContainedBy.getContainers();
    }
    public final Vector getContainers(Class containerClass) {
        return m_isContainedBy.getContainers(containerClass);
    }
}
```

Through the use of these classes, objects can set up the inverse relationships between other objects.

Add New Classes

Next was the creation of several custom classes to add functionality to the object model. These classes would support the functionality that was not in the IFC model, but that were required to support the *Building Composer* application: MDSActivity, MDSFunction, MDSFunctionTemplate, MDSFunctionInstance, MDSProgram (helper interface Int_ProgramContainer), MDSProperty, MDSRequirement, MDSAttribute, and MDSRequirementSet (helper class MDSRequirementSetContainer and interface Int_MDSRequirement-SetContainer). An important concern of these new classes was determining how to output these classes and their data in an IFC output file (or if it should be). The dumpToIFC method defined in the MDSAppInst class had to be overridden in most of the following classes to address this concern.

MDSFunction, MDSFunctionTemplate, and MDSFunctionInstance

The creation of these classes was necessary because the IFC model did not adequately represent the concept of a template (or set of default values) for a functional area. This template would be added to the project and would then be used for the creation of function instances. The function template is a local project copy of function templates defined in an external database. Requirement values from the template are used as the default values during the creation of a function instance. Following the creation of function instances, spaces are created from the function instance. An important concept is that there is only one function template within the project, but there can be many function instances that were created from that function template (Figure 6).

A function can also establish relations to other functions. The first type of relationship is a parent/child relationship. In this relationship one function, the parent, is specialized or refined in another function, the child. An example of this would be the definition of an office function and the specialization in a private office function.

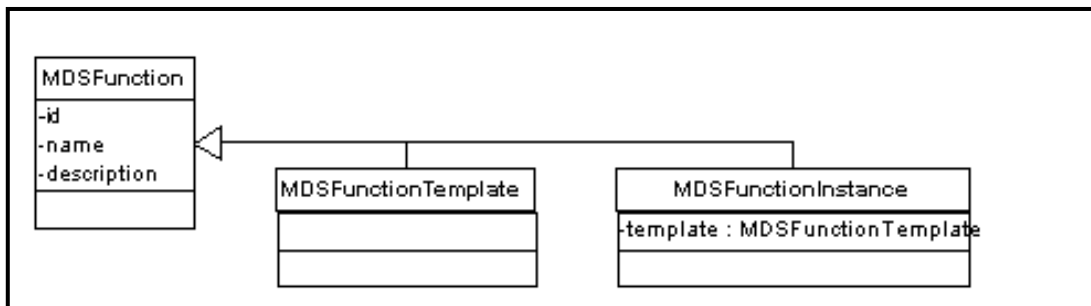


Figure 6. Relation between MDSFunction, MDSFunctionTemplate, and MDSFunctionInstance.

The second type of relationship is a super/sub relationship. In this relationship one function contains another function. This relationship allows for a grouping of functions into a common entity for assignment to a space. An example of this would be the definition of a lobby function, the super, that would contain a reception, waiting, and display functional area, the subs.

Though these relationships are coded in the object model, they are not fully tested and cannot be accessed through the *Building Composer* core library. Further testing and analysis of these relationships is needed.

MDSProgram

The MDSProgram class is used to associate MDSFunctionTemplate and MDSFunctionInstance objects with an MDSProject, MDSSite, MDSBuilding, or MDSBuildingStorey object (its container object). These objects can have a program of function templates, with a planned area, entered by the user and then related to a set of function instances.

To allow multiple objects to reference the same function template, but have individual planned areas, a hashtable was used within MDSProgram. This hashtable would have the function template as a key and then an instance of the helper class MDSFunctionAreaAndSpace as the value. The class MDSFunctionAreaAndSpace would maintain the planned area for that linkage and also have a hashtable. This hashtable would have the function instance as a key and then a vector of spaces as the key.

In **Figure 7**, a site, “Site 1,” and story, “Story 1,” objects are shown with their program objects. There are also two function templates available, “Function Template 1” and “Function Template 2.” From the diagram, it can be seen that the site program hashtable has a reference to “Function Template 1” and that there is a planned area of 1000. This reference has one function instance, “Instance 1,” and two spaces, “Space 1” and “Space 2.” The story program hashtable has a reference to “Function Template 1” and “Function Template 2.” The reference to “Function Template 1” has a planned area of 800. This reference has one function instance, “Instance 2,” and one space, “Space 3.” The reference to “Function Template 2” has a planned area of 500. This reference has two function instances: “Instance 3” has two spaces, “Space 5” and “Space 6,” and “Instance 4” has one space, “Space 4.”

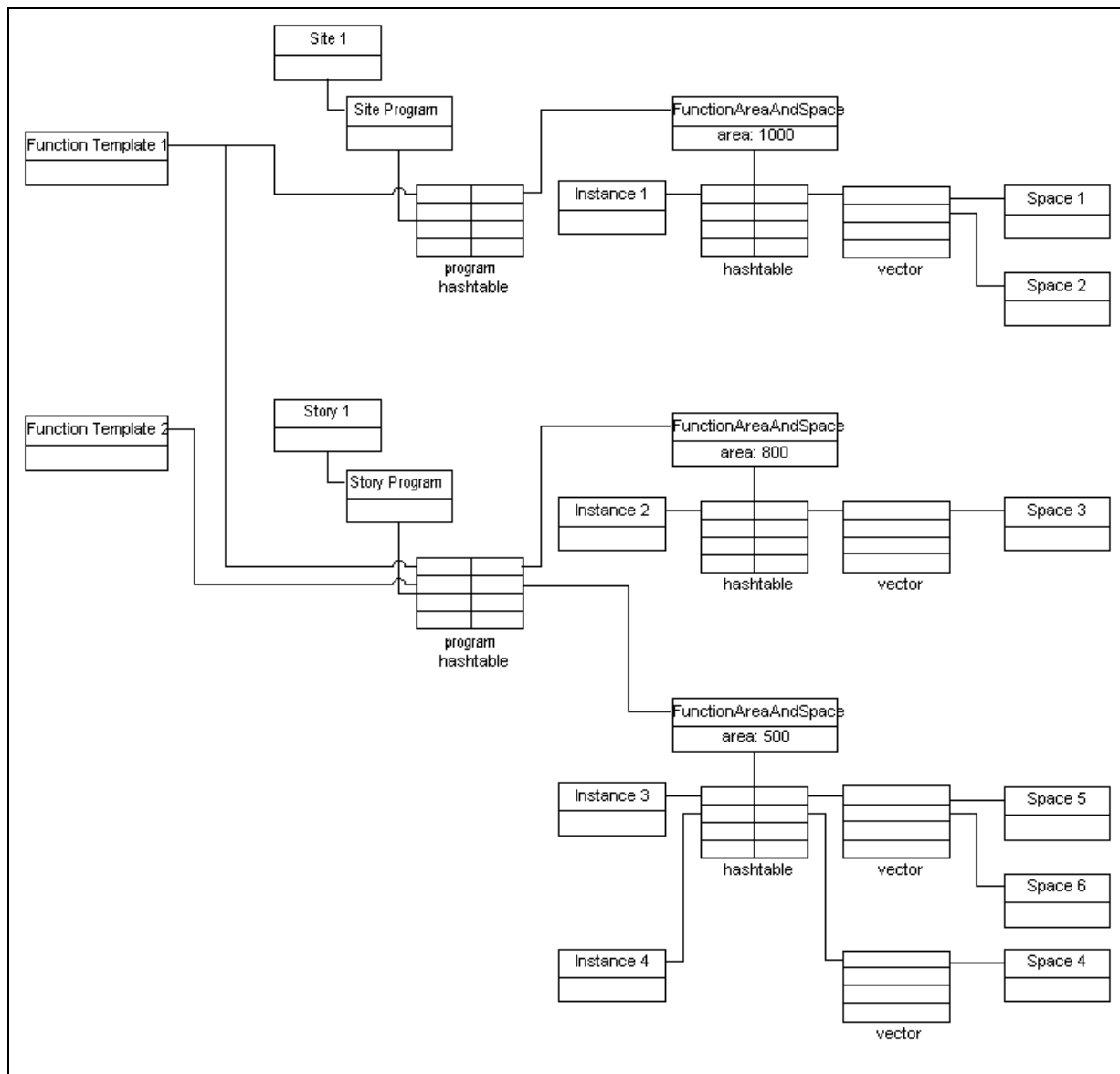


Figure 7. Site and story objects shown with their program objects.

The MDSProgram class contains methods to add/remove function templates, add/remove function instances, add/remove spaces, and setting planned area for a function template. Some methods from the MDSProgram class are:

```

public class MDSProgram extends MDSAppInst {
    // =====
    public void addFunctionTemplate(MDSFunctionTemplate template) {
    public boolean removeFunctionTemplate(MDSFunctionTemplate template) {
    public Vector getFunctionTemplates() {
    public MDSFunctionTemplate findFunctionTemplate(String functionTemplateName) {
    // =====
    public void addFunctionInstance(MDSFunctionInstance instance) {
    public boolean removeFunctionInstance(MDSFunctionInstance instance) {
    public Vector getFunctionInstances() {
    public Vector getFunctionInstances(MDSFunctionTemplate template) {
    // =====
    public void setPlannedArea(MDSFunctionTemplate template, double plannedArea) {
    public double getPlannedArea(MDSFunctionTemplate template) {
  
```

```
// =====
public void addSpace(MDSFunctionInstance instance, MDSSpace space) {
public boolean removeSpace(MDSSpace space) {
public Vector getSpaces(MDSFunctionTemplate template) {
public Vector getSpaces(MDSFunctionInstance instance) {
}
```

The object that would contain the MDSProgram instance would then implement the Int_MDSProgramContainer interface. This interface defined several methods to allow an object to get its program, planned areas, placed areas, function instances, and spaces.

MDSProperty, MDSRequirement, and MDSAttribute

The creation of these classes was for the storage and maintaining of attributes related to object properties. An object can have associated property values attached. These values have a datatype, value, unit of measure, and other attributes with them. **Figure 8** shows the relationship between these classes.

The base MDSProperty class defines most of the attributes and functionality of these classes. Most of the attributes are self-explanatory, but the following describe some of the others:

- **DataType** The datatype for the value. Possible values: 1 = string, 2 = double, 3 = integer, 4 = Boolean.
- **Uom** The unit of measure for the value, i.e., inches, square feet.
- **Source** The justification for the default value.
- **Comment** A user editable field that indicates the reason the user entered a specific value.

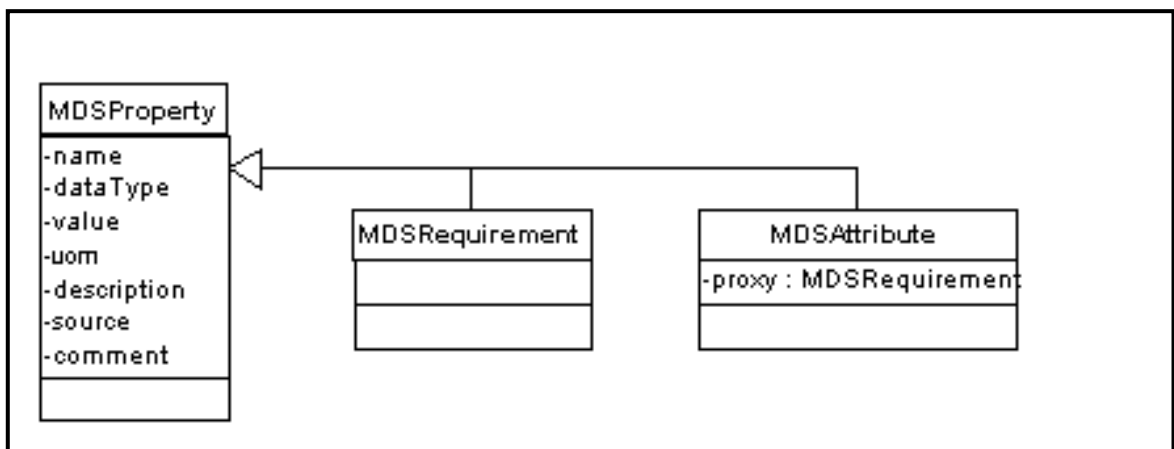


Figure 8. Relationship between MDSProperty, MDSRequirement, and MDSAttribute classes.

Typically an MDSRequirement object is created and added to an object, either a project, site, building, story, function, or space. The MDSAttribute object is used at certain times with a space object. A space object “inherits” the properties of the function instance that defines it. For example, if the function instance has a property “wallColor” with a value of “blue,” then the space should have that property as well. If the function instance changed the value from “blue” to “red” then the new value should be displayed in all spaces of that function instance type. However, the space should be able to override that value if desired. To accomplish this, the property “wallColor” from the function instance is created on the space object as an MDSAttribute object. The proxy attribute for this object would then reference the MDSRequirement object on the function instance. All interactions with the MDSAttribute object then call the proxy object for completion. However, when a value change is attempted on the MDSAttribute object, the reference to the proxy object is removed and the MDSAttribute object deals with all interactions directly.

MDSRequirementSet

The MDSRequirementSet class is used to store the MDSRequirement instances related to an object. An MDSRequirementSet object has a tab name attribute that is used to specify which tab the set should be displayed on. The helper class MDSRequirementSetContainer was created to assist in the relationship between an object and an MDSRequirementSet object. The MDSRequirementSetContainer class is used in the delegation design pattern with other objects.

Database Interface

To provide an independent view to the database being used for persistent storage of the object model, an interface was defined: Int_MDSDatabase. Through the definition of an independent interface, several different backend database systems could be programmed for use. This interface defines the methods required for a database. These methods can be grouped into opening and closing the database, transaction management, root methods, bind methods, migrate object, destroy object, and enumeration of objects.

The opening and closing methods are:

```
/** Opens data storage. Will create database if it does not exist. Will open database in update mode. */
    public int openDatabase(String name);
/** Closes data storage. */
    public boolean closeDatabase();
/** Returns true if database has been opened. */
    public boolean isDatabaseOpen();
```

```
/** Gets name of current database. */
public String getDatabaseName();
```

The transaction management methods are:

```
/** Returns if a transaction is in progress. */
public boolean transactionInProgress();

/**
 * Returns transaction type.
 * 1 = read only
 * 2 = update
 * -1 = no current transaction
 * -999 = error
 */
public int transactionType();

/** Starts an update transaction. */
public boolean startUpdateTransaction();

/** Starts a read-only transaction. */
public boolean startReadOnlyTransaction();

/** Commits the current transaction. */
public boolean commitTransaction();

/** Aborts the current transaction. */
public boolean abortTransaction();
```

The root methods are:

```
/** Creates a root object with the given name if it doesn't exist. The root object is a Vector object. */
public boolean createRoot(String name);

/** Deletes a root object with the given name if it exists. */
public boolean deleteRoot(String name);

/** Gets the root object for name given. The root object is a Vector object. */
public Vector getRoot(String name);

/** Adds an object to the root object for name given. The root object is a Vector object. */
public boolean addToRoot(String name, Object obj);
```

A root is a concept common in many object-oriented databases. A root is a reference to an individual object within the database, an entry point. At least one root object is needed in a database. A single root can be sufficient, but it may be more efficient to have more. In general, every object in the database should not be associated with a root. This is bad for performance. Each root refers to exactly one object.

The bind methods are:

```
/**
 * Binds an object in the database with the given name. The object can later be
 * retrieved with method getBoundObject.
 * Aborts transaction if exception!
 */
/**
 * @param obj = Object to bind in database.
 * @param name = String for bound object.
 */
public boolean bindObject(Object obj, String name);

/**
 * Unbinds object with the given name.
 * Aborts transaction if exception!
 */
/**
 * @param name = String of bound object.
 */
public boolean unbindObject(String name);

/**
 * Gets the bound object with the given name.
 * Aborts transaction if exception!
 */
/**
 * @param name = String of bound object to search for.
 * @return Object with bound name; null otherwise.
```

```

*/
    public Object getBoundObject(String name);
/**
 * Checks for a bound object with the given name.
 * Does NOT abort transaction if exception!
 *
 * @param name = String of bound object to check for.
 */
    public boolean checkBoundObject(String name);

```

The migrate method is:

```

/**
 * Migrates the given object to the database. The object must be persistent-capable.
 *
 * @param obj = Object to migrate to database.
 */
    public boolean migrateObject(Object obj);

```

The destroy method is:

```

/**
 * Destroys the given object from the database.
 * The user is responsible for making sure all references to the destroyed object are cleaned up.
 *
 * @param obj = Object to destroy in database.
 */
    public boolean destroyObject(Object obj);

```

The all object enumeration methods are:

```

/**
 * Populates Vector with all objects in the database. Does not return 'java' package objects.
 * If 'java' objects desired, use allObjects(Vector vec, String name) with 'java' parameter.
 *
 * @param vec = Vector that will be populated. This should be created by calling method.
 */
    public boolean allObjects(Vector vec);
/**
 * Populates Vector with all objects in the database that 'isInstance' for the Class given.
 *
 * @param vec = Vector that will be populated. This should be created by calling method.
 * @param compareClass = Class to compare to.
 */
    public boolean allObjects(Vector vec, Class compareClass);

```

There are two classes created that implement this interface, MDSLocalDb and MDSNoDb. The MDSLocalDb class is for use with the ObjectStore PSE Pro database. The MDSNoDb class will not save any objects persistently. All data is maintained in memory and lost when the application ends.

Controller and Helper Classes

In an effort to try and make the object model easier to use, several additional classes were created. The first of these were the controller classes.

The controller classes consist of MDSModelController, MDSProjectController, MDSObjectLinker, and MDSIFCController.

The helper classes consist of Int_MDSDumpToIFC and MDSObjectExtents.

MDSModelController

The MDSModelController class is responsible for starting, initializing, and stopping a session with the object model. A session creates a context in which you can create transactions, access a database, and manipulate persistent objects within the object-oriented database. This controller maintains a variable that points to the database server that the application has created.

During start of the session, this controller attempts to startup, login, and open the database server.

MDSProjectController

The MDSProjectController class was created to help traversing the major elements of a project easier. These elements are Clfcproject, Clfcsite, Clfcbuilding, and Clfcbuildingstorey. This controller maintains the concept of a current element, set/get of the current element, and ability to get the next type of element in line (get the building elements from a site element). One important aspect of this controller is that it sets the project element to be a root object that can later be retrieved (see section “Database Interface” above about root objects).

```
/**
 * Adds a Clfcproject to the database (if one doesn't already exist).
 * The new project is made the current project.
 */
public boolean addProject() {
    boolean result = false;
    if (projectExists() == true) return result;
    m_database.startUpdateTransaction();
    if (m_database.addToRoot(PROJECT_ROOT_NAME, new Clfcproject(new double[] {0, 0, 0})) == true) {
        setCurrentProject();
        result = true;
    }
    m_database.commitTransaction();
    return result;
}
```

MDSObjectLinker

The MDSObjectLinker controller is a class that is used to help implement the inverse relationship attributes (see section “Implement Inverse Relationship Attributes” above). It contains many methods that help create and access both sides of the one-to-one and one-to-many relationships.

PSE Post-processing

Once the object model classes have been setup, the next step is to prepare them for persistent storage within the object-oriented database. The database used is Objectstore Personal Storage Edition (PSE) Pro from Excelon Corp. PSE uses a post-processor to modify classes to make them persistent capable; i.e., so they can be stored within the database. All classes from the object model must be run through this post-processor.

In addition, two custom classes were created that “wrap” persistent versions of a Hashtable and Vector supplied with PSE. These two classes are MDSHashtable and MDSVector.

The complete listing of the batch file used is in Appendix D. However, the main emphasis is that the process requires several steps that must be executed in order. The first step is to make the Capp_inst class persistent. This class contains two fields that do not need to be stored persistently. This is the reason for the “-tf” option. The second step is a normal post-processing of all classes within the object model. The third step is to make certain helper classes generated from StepTools persistent aware. This is the reason for the “-pa” option. The final step is to make the controller classes persistent aware.

Object Model Use

There are several ways in which the object model can be used. The most basic way is through a simple Java method that creates and manipulates the objects. This approach, however, would provide no persistent storage of the objects created. To persist objects to the database, a transaction must be started, objects created, objects added to the database or referred to by an object already in the database, and then the transaction closed. Here is an example:

```
public class test {
    public static void main(String[] args) {
        try {
            // Create/Initialize database server
            MDSLocalDb mainDatabaseServer = new MDSLocalDb();
            mainDatabaseServer.startup();
            mainDatabaseServer.login("j-heckel");
            mainDatabaseServer.openDatabase("testDb.odb");
            mainDatabaseServer.startUpdateTransaction();
            // Create instances
            Clfcownerhistory owner = new Clfcownerhistory();
            mainDatabaseServer.migrateObject(owner);
            owner.setOwnerdescriptor("Jeff Heckel");
            owner.setApplicationid("SDAITestApplication");
            Clfcpropertytypedef propDef = new Clfcpropertytypedef();
            mainDatabaseServer.migrateObject(propDef);
            double[] temp1 = { 0., 0., 1. };
            Clfcdirection axis = new Clfcdirection();
            mainDatabaseServer.migrateObject(axis);
        }
    }
}
```



```

axis.setDirectionratios(temp1);
double[] temp2 = { 1., 0., 0. };
Clfcdirection refDir = new Clfcdirection();
mainDatabaseServer.migrateObject(refDir);
refDir.setDirectionratios(temp2);
double[] temp3 = { 10., 10., 0. };
Clfccartesianpoint point = new Clfccartesianpoint();
mainDatabaseServer.migrateObject(point);
point.setCoordinates(temp3);
Clfcaxis2placement3d axis1 = new Clfcaxis2placement3d();
mainDatabaseServer.migrateObject(axis1);
axis1.setLocation(point);
axis1.setAxis(axis);
axis1.setRefdirection(refDir);
Clfcaxis2placement axis2 = new Clfcaxis2placement();
mainDatabaseServer.migrateObject(axis2);
axis2.setlfaxis2placement3d(axis1);
Clfcllocalplacement place = new Clfcllocalplacement();
mainDatabaseServer.migrateObject(place);
place.setRelativeplacement(axis2);
Clfcwall wall1 = new Clfcwall();
mainDatabaseServer.migrateObject(wall1);
wall1.setOwnerhistory(owner);
wall1.setProjectid("wall1");
wall1.setLocalplacement(place);
mainDatabaseServer.commitTransaction();
mainDatabaseServer.closeDatabase();
}
catch (Exception e) {
    e.printStackTrace();
}
}
}

```

This example migrates each object to the database as created. The problem with this, however, is that there is no way to later retrieve these objects from a single starting point. As mentioned about database roots above (see “Database Interface,” p 35), a root links a string value with an object so that it can later be retrieved. A better approach would be:

```

public class test {
    public static void main(String[] args) {
        try {
            // Create/Initialize database server
            MDSLocalDb mainDatabaseServer = new MDSLocalDb();
            mainDatabaseServer.startup();
            mainDatabaseServer.login("j-heckel");
            mainDatabaseServer.openDatabase("testDb.odb");
            mainDatabaseServer.startUpdateTransaction();
            // Create instances
            Clfcownerhistory owner = new Clfcownerhistory();
            owner.setOwnerdescriptor("Jeff Heckel");
            owner.setApplicationid("SDAI TestApplication");
            Clfcpropertytypedef propDef = new Clfcpropertytypedef();
            double[] temp1 = { 0., 0., 1. };
            Clfcdirection axis = new Clfcdirection();
            axis.setDirectionratios(temp1);
            double[] temp2 = { 1., 0., 0. };
            Clfcdirection refDir = new Clfcdirection();
            refDir.setDirectionratios(temp2);
            double[] temp3 = { 10., 10., 0. };
            Clfccartesianpoint point = new Clfccartesianpoint();
            point.setCoordinates(temp3);
            Clfcaxis2placement3d axis1 = new Clfcaxis2placement3d();
            axis1.setLocation(point);
            axis1.setAxis(axis);
            axis1.setRefdirection(refDir);
            Clfcaxis2placement axis2 = new Clfcaxis2placement();
            axis2.setlfaxis2placement3d(axis1);
            Clfcllocalplacement place = new Clfcllocalplacement();
            place.setRelativeplacement(axis2);

```

```

        Clfcwall wall1 = new Clfcwall();
        mainDatabaseServer.bindObject(wall1, "Wall1");
        wall1.setOwnerhistory(owner);
        wall1.setProjectid("wall1");
        wall1.setLocalplacement(place);
        mainDatabaseServer.commitTransaction();
        mainDatabaseServer.closeDatabase();
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}

```

The difference here is that the wall object is bound to the string "Wall1." Since the wall object contains references (either directly or indirectly) to all other objects created, these objects will be added to the database when the wall object is persisted to the database. This is called transitive persistence. Later the wall object can be retrieved with the call:

```

Object obj = mainDatabaseServer.getBoundObject("Wall1");
if (obj instanceof Clfcwall) {
    Clfcwall wall1 = (Clfcwall)obj;
    ...
}

```

The following shows a simplified example more closely related to how a user would interact with the object model. It uses the controller classes to do some of its interactions:

```

// declaration
MDSLocalDb mainDatabaseServer;
MDSModelController mdlCtl;
MDSProjectController prjCtl;
// create database server
mainDatabaseServer = new MDSLocalDb();
// create controllers
mdlCtl = new MDSModelController(mainDatabaseServer);
prjCtl = mdlCtl.getProjectController();
// start session and initialize controllers
try {
    if (mdlCtl.startSession(dbName, "BuildingComposer") == 0) {
        errorMessage("Open Database Error.\nError starting session!");
        return;
    }
    if (mdlCtl.init() == false) {
        errorMessage("Open Database Error.\nError initializing controllers!");
        return;
    }
}
catch (Exception e) {
    errorMessage("Error starting session and initializing controllers.");
    errorMessage(e);
    return;
}
// get project, if does not exist create it
Clfcproject project = prjCtl.getCurrentProject();
if (project == null) {
    if (!prjCtl.addProject()) {
        errorMessage("Unable to add Project!");
        return false;
    }
    project = prjCtl.getCurrentProject();
}
if (project == null) return false;
// get site, if does not exist create it
Clfcsite site = prjCtl.getCurrentSite();
if (site == null) {

```

```
        if (!prjCtl.addSite()) {
            errorMessage("Unable to add Site!");
            return false;
        }
        site = prjCtl.getCurrentSite();
    }
    if (site == null) return false;
    // start an update transaction, create elements, and close transaction
    mainDatabaseServer.startUpdateTransaction();
    Clfcownerhistory history = new Clfcownerhistory()
    history.setOwnerdescriptor("The Main Application");
    project.setOwnerhistory(history);
    mainDatabaseServer.commitTransaction();
    // stop session
    if (mdlCtl.isSessionStarted()) {
        mdlCtl.stopSession();
    }
}
```

4 *Building Composer* Core Library

The *Building Composer* core library is separated into several packages to help isolate code and make it more reusable for final application development. This core library is used by the Criteria Composer application to provide most of its functionality and graphical elements. This core library was also used in the development of the Layout Composer application and because of its reusability saved a significant amount of development time. The following will describe several of these packages and point out some key concepts and components of core library.

The *Building Composer* core library requires two environment variables for proper operation. The first variable is “BuildingComposer.” This variable should point to the main install directory for *Building Composer*. This directory is where Java property files and the main library file will be looked for. The second variable is “BuildingComposer_Project.” This variable will indicate the directory that *Building Composer* will create and search for projects.

Common Graphical Components and Utilities

Several common graphical user component classes were created that can be used in the creation of display panels. These common components are for text fields, pull-down fields, list fields, label fields, and horizontal and vertical panels. These components can be used within the JBuilder designer mode when creating JPanel displays. These component classes are in package BCGUI.GUIBeans.

In the package BCGUI.Utills, there are several classes used throughout Criteria Composer. Some of the main classes of importance are BaseLibrary and MainLibrary; ActivityLibrary, RequirementSetLibrary, PropertyLibrary, and FunctionLibrary; DataNode; and JDBCDBCAccess. Some of these are described below.

Library Classes

There are several classes in the BCGUI.Utills package that define libraries of information that are searched throughout the running of the *Building Composer* core library. These classes are ActivityLibrary (store MDSActivity objects), Re-

requirementSetLibrary (store MDSRequirementSet objects), PropertyLibrary (store MDSProperty objects), and FunctionLibrary (store MDSFunction objects). These stores of information are populated at the start of the *Building Composer* core library by the MainLibrary class. This class reads a table row from an external database (or datasource), creates an object, and then stores it in a library. These libraries are then searched when certain actions occur. An example is when a new building object is created. The RequirementSetLibrary would be searched to see if any requirements should be added to this new building object. Currently a Microsoft Access relational database is used as the external database (see Appendix E for a listing of the required tables). Here is the code that reads and creates entries for the ActivityLibrary:

```
private boolean readActivityLibrary() {
    // read activity values
    try {
        JDBCDBCAccess dataSource = new JDBCDBCAccess(m_dataSourceName, "select * from ActivityHeaders");
        Vector name = dataSource.populateVector(1);
        Vector description = dataSource.populateVector(2);
        for (int i = 0; i < name.size(); ++i) {
            String activityName = (String)name.elementAt(i);
            MDSActivity activity = m_activityLibrary.findActivity(activityName);
            if (activity == null) {
                activity = new MDSActivity(activityName);
                activity.setDescription((String)description.elementAt(i));
                m_activityLibrary.addActivity(activity);
            }
        }
    }
    catch (Exception exc) {
        BuildingComposer.errorMessage(exc);
        return false;
    }
    return true;
}
```

See Appendix F for further information on the creation and use of these libraries.

DataNode Class

The DataNode class defines the elements of data maintained within a project tree (see “Project Tree” p 45). It is a class that can maintain a reference to a minimum of one piece, and a maximum of three pieces, of data. A text string is attached to the main piece of data and returned from the toString method.

```
public class DataNode {
    Object m_data;
    String m_dataLabel;
    Object m_userData1;
    Object m_userData2;
    // =====
    /** Creates a DataNode object with the given data object. */
    public DataNode(Object data, String dataLabel) {
        m_data = data;
        m_dataLabel = dataLabel;
    }
    /** Creates a DataNode object with the given data object plus one additional piece of data. */
    public DataNode(Object data, String dataLabel, Object userData1) {
        this(data, dataLabel);
        m_userData1 = userData1;
    }
}
```

```

/** Creates a DataNode object with the given data object plus two additional pieces of data. */
public DataNode(Object data, String dataLabel, Object userData1, Object userData2) {
    this(data, dataLabel, userData1);
    m_userData2 = userData2;
}
// =====
public Object getData() {
    return m_data;
}
// =====
public void setDataLabel(String label) {
    m_dataLabel = label;
}
public String getDataLabel() {
    return m_dataLabel;
}
// =====
public void setUserData1(Object userData1) {
    m_userData1 = userData1;
}
public Object getUserData1() {
    return m_userData1;
}
// =====
public void setUserData2(Object userData2) {
    m_userData2 = userData2;
}
public Object getUserData2() {
    return m_userData2;
}
// =====
public String toString() {
    return m_dataLabel;
}
}

```

Project Tree

The main graphical element of a *Building Composer* application is the project tree. This tree displays the main elements within a project: the Project, Site, Building, Story, Function Instance, and Space element. The project tree is responsible for the display of these main project elements and generation of messages when certain actions are performed on the tree. These actions are a key press, a mouse button clicked, a mouse button pressed, a mouse button released, and a tree selection value change.

The base class for a project tree is `BaseProjectTree`. This class registers listeners for the above-described tree actions and maintains the list of listeners to be notified when any of these actions occur. The following are the methods called to register listeners:

```

public synchronized void addProjectTreeListener(ProjectTreeListener listener) {
    public synchronized void removeProjectTreeListener(ProjectTreeListener listener) {

```

The methods that a listener would then implement to receive these messages (see `ProjectTreeListener` interface) are:

```

/*-----
 * ProjectTreeListener methods
 *-----*/
public void projectTree_keyTyped(KeyEvent e) {
public void projectTree_mouseClicked(MouseEvent e) {
public void projectTree_valueChanged(TreeSelectionEvent e) {

```

The main class for a project tree is ProjectTree. This class inherits from BaseProjectTree to properly display the main project elements within the tree. The following breaks down the code of the ProjectTree class:

```

public class ProjectTree extends BaseProjectTree {
// =====
public ProjectTree() {
    super();
    try {
        jblnit();
    }
    catch (Exception e) {
        BuildingComposer.errorMessage(e);
    }
}
private void jblnit() throws Exception {
    // setup empty tree model
    setModel(new DefaultTreeModel(createProjectBranch(null)));
    // create renderer
    DefaultTreeCellRenderer renderer = new DefaultTreeCellRenderer();
    renderer.setClosedIcon(null);
    renderer.setLeafIcon(null);
    renderer.setOpenIcon(null);
    // setup tree
    setCellRenderer(renderer);
    getSelectionModel().setSelectionMode(TreeSelectionModel.SINGLE_TREE_SELECTION);
}
}

```

An internal method to get the “Code” property of an object, often used in the display of an object within the tree is:

```

// =====
private String getLabel(MDSObject obj) {
    try {
        String id = obj.getStringProperty("Code");
        if (id == null) id = new String();
        return id;
    }
    catch (Exception e) {
        BuildingComposer.errorMessage("ProjectTree:Exception getting label: obj: " + obj);
        return new String();
    }
}
}

```

Various methods can create node labels:

```

// =====
/** Creates a project node label. Needs to be called within a transaction. */
public String projectLabel(MDSProject project) {
    return new String("Project " + getLabel(project));
}
public String siteLabel(MDSSite site) {
public String buildingLabel(MDSBuilding building) {
public String storyLabel(MDSBuildingStorey story) {
public String functionLabel(MDSFunctionInstance function) {
public String spaceLabel(MDSSpace space) {

```

Various methods can be used to create a tree node. A node is a `DefaultMutableTreeNode` (see JDK documentation) with a `DataNode` object as a piece of user data (see section “`DataNode Class`” above):

```
// =====
/** Creates a project node. The DataNode contains a MDSProject object as data. Needs to be called within a transaction. */
public DefaultMutableTreeNode projectNode(MDSProject project) {
    return new DefaultMutableTreeNode(new DataNode(project, DataNode.OBJECT, projectLabel(project)));
}
public DefaultMutableTreeNode siteNode(MDSSite site) {
public DefaultMutableTreeNode buildingNode(MDSBuilding building) {
public DefaultMutableTreeNode storyNode(MDSBuildingStorey story) {
public DefaultMutableTreeNode functionNode(MDSProgram program, MDSFunctionTemplate template, MDSFunctionInstance function) {
public DefaultMutableTreeNode spaceNode(MDSFunctionTemplate template, MDSFunctionInstance function, MDSSpace space) {
```

Various methods can create branches within a tree:

```
// =====
/** Need to check for null return value. Needs to be called within a transaction. */
public DefaultMutableTreeNode createProjectBranch(MDSProject project) {
    DefaultMutableTreeNode prjNode = new DefaultMutableTreeNode(new DataNode(null, 0, "Project"));
    if (project == null) return prjNode;
    try {
        // create project node
        prjNode = projectNode(project);
        // site node
        prjNode.add(createSiteBranch(project.getSite()));
        // building nodes
        Enumeration buildings = project.getSite().getBuildings().elements();
        while (buildings.hasMoreElements()) {
            ClfcBuilding bldgObj = (ClfcBuilding)buildings.nextElement();
            prjNode.add(createBuildingBranch(bldgObj));
        }
    }
    catch (Exception e) {
        BuildingComposer.errorMessage("ProjectTree:Exception building project branch!");
        BuildingComposer.errorMessage(e);
    }
    return prjNode;
}
public DefaultMutableTreeNode createSiteBranch(MDSSite site) {
public DefaultMutableTreeNode createBuildingBranch(MDSBuilding building) {
public DefaultMutableTreeNode createStoryBranch(MDSBuildingStorey story) {
public DefaultMutableTreeNode createFunctionBranch(MDSProgram program, MDSFunctionInstance function) {
public DefaultMutableTreeNode createSpaceBranch(MDSFunctionInstance function, MDSSpace space) {
```

The following method is overridden to display the main project elements:

```
// =====
public void reset() {
    super.reset();
    if (BuildingComposer.mainDatabaseServer.isDatabaseOpen()) {
        BuildingComposer.mainDatabaseServer.startReadOnlyTransaction();
        setModel(new DefaultTreeModel(createProjectBranch(BuildingComposer.prjCtl.getCurrentProject())));
        BuildingComposer.mainDatabaseServer.commitTransaction();
    }
    else {
        setModel(new DefaultTreeModel(createProjectBranch(null)));
    }
    expandRow(0);
    setSelectionRow(1); // causes TreeSelectionEvent - select site object
    setSelectionRow(0); // causes TreeSelectionEvent - select project object
    setShowsRootHandles(true);
}
}
```


The image shows a software interface titled "Design Criteria". It has a tabbed menu with "Program", "Properties", "Architecture", "Electrical", "HVAC", "Lighting", "Plumbing", and "Security". The "Properties" tab is selected. Below the tabs is a section titled "Project Properties" containing several input fields: "Code:" with the value "BC1", "Name:" with the value "Barracks Complex", "City:" with the value "Dallas", "State:" with a dropdown menu showing "TX", "Zip:" with an empty field, "Number:" with an empty field, "Fiscal:" with an empty field, and "Units:" with a dropdown menu showing "English".

Figure 9. Property panel classes.

Property Panels

The property panel classes (Figure 9) are used to display information relevant to an object that is not displayed within a requirement grid (see section “Requirement Grid” below). The PropPanel class is the abstract base class for most of the other classes: ProjectPropPanel, SitePropPanel, BuildingPropPanel, StoryPropPanel, FunctionPropPanel, and SpacePropPanel. A property panel is typically made up of common graphical components described above (see section “Common Graphical Components and Utilities” above). When the value of these components is updated, event messages are generated and sent to registered listeners.

Here are some code highlights of PropPanel:

```
public abstract class PropPanel extends JPanel {
    // =====
    // Method to try and convert the text value of the given FieldRow into a double.
    protected double convertToDouble(FieldRow row);
    // =====
    // Method to update the given propertyName in the given obj with the text value from the given row.
    protected boolean updateFromFieldRow(MDSObject obj, String propertyName, FieldRow row);
    // =====
    // Method to update the given propertyName in the given obj with the text value from the given row.
    protected boolean updateFromComboBoxRow(MDSObject obj, String propertyName, ComboBoxRow row);
}
```

The following methods add and remove listeners, and enable the event messaging from the property panels:

```
/** Method to add a listener for property panel change events. */
public synchronized void addPropertyPanelListener(PropertyPanelListener listener);
/** Method to remove a listener for property panel change events. */
public synchronized void removePropertyPanelListener(PropertyPanelListener listener);
// Method to generate a property panel change event.
protected void notifyPropertyChanged(MDSObject obj, String property, Object value);
}
```

A listener would then implement these methods to receive property change messages (see PropertyPanelListener interface):

```
public void projectPropertyChange(MDSProject project, String property, Object value);
public void sitePropertyChange(MDSSite site, String property, Object value);
public void buildingPropertyChange(MDSBuilding building, String property, Object value);
public void storyPropertyChange(MDSBuildingStorey story, String property, Object value);
public void functionPropertyChange(MDSFunction function, String property, Object value);
public void spacePropertyChange(MDSSpace space, String property, Object value);
```

A typical property panel then maintains a pointer to its current object and populates text fields with appropriate values. A typical example of ProjectPropPanel is:

```
public class ProjectPropPanel extends PropPanel {
```

The current object is:

```
MDSProject curProject;
ProjectTree curTree = null;
BorderLayout borderLayout1 = new BorderLayout();
```

The panel graphical elements are:

```
VerticalPanel jPanel2 = new VerticalPanel();
FieldRow idRow = new FieldRow("ID:", "ID of the selected Project");
FieldRow codeRow = new FieldRow("Code:", 8, "Code of the selected Project.");
FieldRow titleRow = new FieldRow("Name:", "Name of the selected Project.");
HorizontalPanel jPanel3 = new HorizontalPanel();
FieldRow cityRow = new FieldRow("City:", "City in which the selected Project is located.");
FieldRow stateRow = new FieldRow("State:", 2, "State in which the selected Project is located.");
FieldRow zipRow = new FieldRow("Zip:", 10, "Zip code in which the selected Project is located.");
HorizontalPanel jPanel4 = new HorizontalPanel();
FieldRow numberRow = new FieldRow("Number:", 8, "Number of the selected Project.");
FieldRow fiscalRow = new FieldRow("Fiscal:", 12, "Fiscal Year of the selected Project.");
FieldRow unitsRow = new FieldRow("Units:", "Units of the selected Project.");
// =====
public ProjectPropPanel() {
    try {
        jblInit();
    }
    catch (Exception ex) {
        BuildingComposer.errorMessage(ex);
    }
}
// =====
private void jblInit() throws Exception {
```

To set up the panels:

```
this.setLayout(borderLayout1);
```

```

// set glues and strut sizes
codeRow.setAddEndGlue(true);
codeRow.setStrutSize(25);
titleRow.setStrutSize(21);
cityRow.setStrutSize(35);
DocumentFilter stateDoc = new DocumentFilter(2, " ");
stateDoc.setUpperCaseOnly(true);
stateRow.getTextField().setDocument(stateDoc);
DocumentFilter zipDoc = new DocumentFilter(10, " ,/,<>?{}|[]\\t+=~!@#%&^&*()_.;:~\"");
zipDoc.setNoUpperCase(true);
zipDoc.setNoLowerCase(true);
zipRow.getTextField().setDocument(zipDoc);
jPanel2.setTitleString("Project Properties");
jPanel2.setBorderType(2);
if (BuildingComposer.developerVersion) jPanel2.add(idRow);
idRow.setEnabled(false);
jPanel2.add(codeRow);
jPanel2.add(titleRow);
jPanel3.add(cityRow);
jPanel3.add(stateRow);
jPanel3.add(zipRow);
jPanel2.add(jPanel3);
jPanel4.add(numberRow);
jPanel4.add(fiscalRow);
jPanel4.add(unitsRow);
unitsRow.setEnabled(false);
jPanel2.add(jPanel4);

```

Add focus and action listeners for each editable field. These listeners will start the event message generation and sending:

```

codeRow.getTextField().addFocusListener(new java.awt.event.FocusAdapter() {
    public void focusLost(FocusEvent e) {
        if (updateFromFieldRow("Code", codeRow)) {
            updateTreeNodeLabel();
        }
    }
});
codeRow.getTextField().addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        if (updateFromFieldRow("Code", codeRow)) {
            updateTreeNodeLabel();
        }
    }
});
titleRow.getTextField().addFocusListener(new java.awt.event.FocusAdapter() {
    public void focusLost(FocusEvent e) {
        updateFromFieldRow("Name", titleRow);
    }
});
titleRow.getTextField().addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        updateFromFieldRow("Name", titleRow);
    }
});
cityRow.getTextField().addFocusListener(new java.awt.event.FocusAdapter() {
    public void focusLost(FocusEvent e) {
        updateFromFieldRow("City", cityRow);
    }
});
cityRow.getTextField().addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        updateFromFieldRow("City", cityRow);
    }
});
stateRow.getTextField().addFocusListener(new java.awt.event.FocusAdapter() {
    public void focusLost(FocusEvent e) {
        updateFromFieldRow("State", stateRow);
    }
});
stateRow.getTextField().addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        updateFromFieldRow("State", stateRow);
    }
});

```

```

});
zipRow.getTextField().addFocusListener(new java.awt.event.FocusAdapter() {
    public void focusLost(FocusEvent e) {
        updateFromFieldRow("Zip", zipRow);
    }
});
zipRow.getTextField().addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        updateFromFieldRow("Zip", zipRow);
    }
});
numberRow.getTextField().addFocusListener(new java.awt.event.FocusAdapter() {
    public void focusLost(FocusEvent e) {
        updateFromFieldRow("Number", numberRow);
    }
});
numberRow.getTextField().addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        updateFromFieldRow("Number", numberRow);
    }
});
fiscalRow.getTextField().addFocusListener(new java.awt.event.FocusAdapter() {
    public void focusLost(FocusEvent e) {
        updateFromFieldRow("FY", fiscalRow);
    }
});
fiscalRow.getTextField().addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        updateFromFieldRow("FY", fiscalRow);
    }
});
this.add(jPanel2, BorderLayout.CENTER);
}

```

An update of certain fields may require a display update of the main project tree:

```

// =====
private void updateTreeNodeLabel() {
    if (curTree != null) {
        DefaultMutableTreeNode treeNode = curTree.getSelectedNode();
        DataNode dataNode = (DataNode)treeNode.getUserObject();
        BuildingComposer.mainDatabaseServer.startReadOnlyTransaction();
        dataNode.setDataLabel(curTree.projectLabel(curProject));
        BuildingComposer.mainDatabaseServer.commitTransaction();
        curTree.updateSelectedNode(dataNode);
    }
}

```

The following is called to set the main project tree (may need a display update):

```

// =====
public void setProjectTree(Object tree) {
    if (tree instanceof ProjectTree) {
        curTree = (ProjectTree)tree;
    }
    else {
        curTree = null;
    }
}

```

The following is then called to set the current object and fill the field values:

```

// =====
/** Needs to be called within a read-only transaction! */
public void fillPanel(MDSProject project) {
    if (project == null) return;
    curProject = project;
    idRow.getTextField().setText(curProject.getGUID());
    codeRow.getTextField().setText(curProject.getStringProperty("Code"));
    titleRow.getTextField().setText(curProject.getStringProperty("Name"));
    cityRow.getTextField().setText(curProject.getStringProperty("City"));
}

```

```

stateRow.getTextField().setText(curProject.getStringProperty("State"));
zipRow.getTextField().setText(curProject.getStringProperty("Zip"));
numberRow.getTextField().setText(curProject.getStringProperty("Number"));
fiscalRow.getTextField().setText(curProject.getStringProperty("FY"));
if (BuildingComposer.englishUnits == true) unitsRow.getTextField().setText("English");
else unitsRow.getTextField().setText("Metric");
this.revalidate();
this.repaint();
}

```

This is overridden to pass the current object:

```

// =====
private boolean updateFromFieldRow(String propertyName, FieldRow row) {
    return super.updateFromFieldRow(curProject, propertyName, row);
}

```

Requirements

A requirement (or property) in *Building Composer* is implemented using the MDSRequirement class from the object model. Several attributes defined in the MDSRequirement class are used during the *Building Composer* application and are worth mentioning.

A requirement in *Building Composer* has a level attribute attached to it. This integer attribute indicates which object in the hierarchy should have this requirement added to it. The possible values for the level attribute are: 1 = project, 2 = site, 4 = building, 8 = story, 16 = function, 32 = space. These values can be added to have a requirement added at multiple levels, i.e., level = 48 equates to function and space. As objects are created in *Building Composer*, requirements are searched and added as indicated by the level attribute.

A requirement also has an attached isRequired attribute. This Boolean attribute indicates whether a requirement should be automatically added to an object when it is created, e.g., if a requirement is level 4 and isRequired is false, then when a new building object is created, this requirement would not be added.

Requirement Grid

The requirement grid is used to display the requirement sets and requirements for the currently selected object in the project tree. **Figure 10** shows a grid of cells representing a series of requirement sets and their requirements on a specific tab. In this example the tab name is "Architecture," one requirement set name is "1: Size – Horizontal," and one requirement name is "Length Desired."

Design Criteria

Program | Properties | **Architecture** | Electrical | HVAC | Lighting | Plumbing | Communication | Security

Architecture Criteria

Requirement	Value	UOM	Source	Comment
1: Size - Horizontal				
Aspect Ratio Desired	[Not Set]	Ratio (length/width)	Design Guide	
Length Desired	[Not Set]	Feet	Design Guide	
Width Desired	[Not Set]	Feet	Design Guide	
2: Size - Vertical				
Ceiling Height Desired	[Not Set]	Feet	Design Guide	
Floor Thickness Desired	[Not Set]	Feet	Design Guide	
3: Miscellaneous				
Solution for Architecture	[Not Set]	String	System	
Solution for Furniture	[Not Set]	String	System	
Solution for Lighting	[Not Set]	String	System	
Solution for Mechanical	[Not Set]	String	System	
Solution for Plumbing	[Not Set]	String	System	
Solution for Power	[Not Set]	String	System	

Add User Criteria... Add Library Criteria...

Figure 10. Requirement set.

The first class used with the requirement grid is the RequirementGridDataSource class. This class defines the model of how the data displayed is stored. Separating the data model in this way allows for the possibility of different actual displays of the data. This separation follows the model-view-controller design pattern. The RequirementGridDataSource class uses a Vector of Vector objects to cache the data within the grid. Data from requirement sets and requirements, which require database access and are therefore slower, are stored in this data cache making future retrievals faster.

```
public abstract class RequirementGridDataSource implements Int_PropertyGrid, PropertyChangeListener {
    // elements of data in m_data are ListNode objects with either an
    // MDSRequirementSet object or MDSPROPERTY object (with their MDSRequirementSet as user data)
    private Vector m_data = new Vector();
    // data cache for quick access
    private VectorDataCache m_dataCache = new VectorDataCache();
    public void addRequirementSet(MDSRequirementSet reqSet) {
        m_data.addElement(new ListNode(reqSet, reqSet.getName()));
        Enumeration reqs = reqSet.getRequirements().elements();
        while (reqs.hasMoreElements()) {
            MDSPROPERTY property = (MDSPROPERTY)reqs.nextElement();
            m_data.addElement(new ListNode(property, property.getName(), reqSet));
        }
        sortData();
        resetCache();
    }
}
```

Method to get the data item for a specific cell (note the check of the data cache at the beginning and populate of the data cache at the end):

```
public Object getTableDataItem(int row, int col) {
    if (row < 0) return null;
    // use m_dataCache item if present
    Object result = m_dataCache.getItem(row, col);
    if (result != null)
        return result;
    Object obj = getDataElement(row);
    // data, is it a MDSRequirementSet object?
    if (obj instanceof MDSRequirementSet) {
        BuildingComposer.mainDatabaseServer.startReadOnlyTransaction();
        MDSRequirementSet reqSet = (MDSRequirementSet)obj;
        if (col == levelCol()) result = new Integer(-1);
        else if (col == requirementCol()) {
            result = reqSet.getName();
        }
        else if (col == requiredCol()) result = new Boolean(true);
        else if (col == changeCol()) result = new Boolean(false);
        else if (col == thirdPartyCol()) result = reqSet.getLibraryName();
        else if (col == descriptionCol()) result = reqSet.getDescription();
        else if (col == guidCol()) result = reqSet.getGUID();
        else result = new String();
        BuildingComposer.mainDatabaseServer.commitTransaction();
    }
    // data, is it a MDSPProperty object?
    else if (obj instanceof MDSPProperty) {
        BuildingComposer.mainDatabaseServer.startReadOnlyTransaction();
        MDSPProperty property = (MDSPProperty)obj;
        if (col == levelCol()) result = new Integer(property.getLevel());
        else if (col == requirementCol()) result = new String(" " + property.getDisplayName());
        else if (col == requiredCol()) result = new Boolean(property.getIsRequired());
        else if (col == changeCol()) result = new Boolean(property.getValueCanBeChanged());
        else if (col == valueCol()) {
            String data = property.getValue();
            if (data.equalsIgnoreCase("true") || data.equalsIgnoreCase("false")) {
                result = new Boolean(data);
            }
            else {
                result = data;
            }
        }
        else if (col == uomCol()) result = property.getUnitOfMeasure();
        else if (col == sourceCol()) result = property.getSource();
        else if (col == commentCol()) result = property.getComment();
        else if (col == thirdPartyCol()) result = property.getLibraryName();
        else if (col == descriptionCol()) result = property.getDescription();
        else if (col == guidCol()) result = property.getGUID();
        else if (col == nameCol()) result = property.getName();
        BuildingComposer.mainDatabaseServer.commitTransaction();
    }
    // default
    else if (col == requirementCol() || col == valueCol())
        result = obj;
    // add item to m_dataCache
    m_dataCache.addItem(result, row, col);
    return result;
}
```

The RequirementGridDataSource class also has methods to determine if a grid cell is editable and resizable:

```
public boolean isEditable(int row, int col) {
    public boolean isResizable(int col) {
```

Requirement values may require a special editor to be used to update of their values. This typically occurs when the requirement is defined from an extension (see “Extensions,” p 56) that has special validation rules to be checked or value

that cannot adequately be typed in by the user. To accommodate this special editor, two methods are defined:

```
public boolean useSpecialEditor(int row) {
public void specialEdit(int row) {
```

These methods would then be called from within the table that displays the grid before an actual cell edit takes place. For example (here `m_model` is an instance of the `RequirementGridDataSource` class):

```
// stop editing before it happens
public boolean editCellAt(int row, int column, EventObject e){
    // does this cell require a special editor?
    if (column == m_model.valueCol()) {
        if (m_model.useSpecialEditor(row)) {
            m_model.specialEdit(row);
            return false;
        }
    }
    // else...
    return super.editCellAt(row, column, e);
}
```

The requirement grid was implemented using both the Java Swing `JTable` class and the JClass `JCTable` class. This was done because of the limitation of the MicroStation JVM, which was based on Sun JVM 1.1. The JClass `JCTable` class is based on Sun JVM 1.2.

Implementing the requirement grid using the Java Swing `JTable` class is done with the `RequirementGrid`, `RequirementGridTable` (which extends `JTable`), `ReqGridDataSource` (which extends `RequirementGridDataSource`), and `UserTableModel` (which extends `ReqGridDataSource`) classes.

```
class RequirementGrid implements Int_RequirementGrid {
    ReqGridDataSource m_dataSource;
    RequirementGridTable m_dataTable;
class RequirementGridTable extends JTable {
```

Implementing the requirement grid using the JClass `JCTable` class is done with the `RequirementGrid_CC`, `RequirementGridTable_CC` (extends `JCTable`), `ReqGridDataSource_CC` (extends `RequirementGridDataSource`), and `UserTableModel_CC` (extends `ReqGridDataSource_CC`) classes.

```
class RequirementGrid_CC implements Int_RequirementGrid {
    ReqGridDataSource_CC m_dataSource;
    RequirementGridTable_CC m_dataTable;
class RequirementGridTable_CC extends JCTable {
```

Classes `RequirementGrid` and `RequirementGrid_CC` implement interface `Int_RequirementGrid`. This is done so that an instance of either can be created and then used in the same manner. This code from the `TabPanel` class shows how to create a requirement grid based on the Java Version:


```

// create requirement grid
Int_RequirementGrid requirementGrid = null;
try {
    if (<Java Version 1.2>) { // earlier than 1.2 Java Version
        Class reqCl = Class.forName("BCGUI.Panel.DesignCriteria.RequirementGrid");
        requirementGrid = (Int_RequirementGrid)reqCl.newInstance();
    }
    else { Java Version 1.2 or later
        Class reqCl = Class.forName("BCGUI.Panel.DesignCriteria.RequirementGrid_CC");
        requirementGrid = (Int_RequirementGrid)reqCl.newInstance();
    }
}
catch (Exception ex) {
}

```

The Int_RequirementGrid interface is:

```

public interface Int_RequirementGrid {
    /** Sets whether this grid can be edited. The grid will be empty after this call! */
    public void setEditable(boolean editable);
    /** Sets whether this grid can be selectable. */
    public void setSelectable(boolean selectable);
    /** Returns the JComponent to display the grid. */
    public JComponent getDataTable();
    /** Removes all MDSRequirementSets (and their MDSPROPERTY objects) from the grid. */
    public void cleanup();
    /** Adds a MDSRequirementSet (and it's MDSPROPERTY objects) to the grid. */
    public void addRequirementSet(MDSRequirementSet reqSet);
    /** Returns the number of rows in the grid. */
    public int getNumberOfRows();
    /** Gets the currently selected MDSPROPERTY objects from the grid. */
    public Vector getSelected();
}

```

Extensions

Though the *Building Composer* core library has a default set of requirements that are associated with an element, it is not possible that all requirements can be defined. In addition, a future custom application from a third party may need requirements of their own added to perform some type of action or analysis. An extension is the term for one of these custom applications. An extension provides new capabilities and usually requirements to a project. The Layout Composer application is an example of an extension. To become an extension and interact with the *Building Composer* core library, an extension add-on must implement an interface that provides a common entry point for the *Building Composer* core library. This interface is the ExtensionInterface:

```

// Interface that all third party classes must implement.
public interface ExtensionInterface {
    /** Called to get extension name. */
    public String getName();
    /** Called to get extension datasource name. */
    public String getDataSource();
    /** Called to get extension icon or null if none. */
    public Icon getIcon();
    /** Return true if special editor is to be used for given MDSPROPERTY. */
    public boolean isSpecialEditorRequired(MDSPROPERTY property);
    /** Called to allow extension to edit given MDSPROPERTY. */
    public void editRequirement(Frame frame, MDSAppInst obj, MDSRequirementSet requirementSet, MDSPROPERTY property);
    /** Called to allow extension to validate new value for given MDSPROPERTY. */
    public boolean validateValue(MDSAppInst obj, MDSRequirementSet requirementSet, MDSPROPERTY property, String newValue);
}

```

```

/** Return JPanel to display for given MDSRequirementSet. Return null if none. */
public JPanel getCustomPanel(Int_MDSRequirementSetContainer obj, MDSRequirementSet requirementSet);
/** Called to allow extension to set background/foreground colors and font of a property cell. */
public void setPropertyCellStyle(Int_PropertyGrid grid, MDSApplnst obj, MDSRequirementSet requirementSet, MDSProperty property);
}

```

An extension can be loaded manually by the user or automatically at creation of a new project. To have the extension automatically loaded, an entry needs to be made in the BCExtensions.properties file. This entry indicates the name of a component file and the location of that component file. Here is a sample properties file entry would be:

```
LayoutComposer.cmp=d:\\code\\java\\layoutcomposer\\layoutcomposer.cmp
```

The component file will then have an entry that will indicate the archive file to search and the name of the class that implements the ExtensionInterface interface. Here is a sample component file:

```
LayoutComposer.zip;LayoutComposer.LayoutComposerExt
```

The *Building Composer* core library can use this information to try to load the extension. Once loaded, the extension database of default requirements is searched and requirements are added to appropriate elements within the current project. Future elements created will have requirements from the extension database automatically added.

Common Commands

Several developed commands can be used within a final application. The commands developed in the *Building Composer* core library implement the command design pattern. This design pattern allows for easy implementation of an undo and redo capability within an application. Within the *Building Composer* core library, there are primary and sub commands. A sub command does the main piece of work done within a command. Here is the interface that a sub command implements:

```

x/** Command interface based on Command pattern. */
public interface Command {
    /**
     * Perform the command encapsulated by this object.
     * @return <code>true</code> if successful and can be undone.
     */
    * <p>
    * To create single-shot commands, maintain boolean member variable and
    * return false if the object's dolt command was previously called. This
    * variable would be reset upon a successful undolt call.
    */
    public abstract boolean dolt();
    /**
     * Undo the last invocation of dolt.
     * @return <code>true</code> if the undo was successful.
     */
    public abstract boolean undolt();
}

```

```
}
```

Here is an example of a sub command:

```
public class Sub_AddBuilding implements Command {
    private boolean m_Done = false;
    private Clfcsite m_curSite;
    private Clfcbldg m_curBuilding;
    private String m_prjGUID;
    private String m_bldgGUID;
    // =====
    public Sub_AddBuilding(Clfcsite site) {
        BuildingComposer.mainDatabaseServer.startReadOnlyTransaction();
        m_prjGUID = BuildingComposer.prjCtl.getCurrentProject().getGUID();
        BuildingComposer.mainDatabaseServer.commitTransaction();
        m_curSite = site;
    }
    // =====
    public boolean doIt() {
        // if this command has already been done then don't doIt again
        if (m_Done) {
            return false;
        }
        try {
            BuildingComposer.mainDatabaseServer.startUpdateTransaction();
            if (m_curBuilding == null) { // first time running
                // create new building, set current building, get/set guid, and add properties
                BuildingComposer.prjCtl.addBuilding();
                m_curBuilding = BuildingComposer.prjCtl.getCurrentBuilding();
                m_bldgGUID = m_curBuilding.getGUID();
                ExtensionUtilities extUtils = new ExtensionUtilities();
                extUtils.addProperties(m_curBuilding); // add required properties
            }
            else {
                // read previously created building
                // add link
                m_curSite.linkContaine(m_curBuilding);
            }
            BuildingComposer.mainDatabaseServer.commitTransaction();
            // success
            m_Done = true;
            return true;
        }
        catch (Exception e) {
            BuildingComposer.errorMessage(e);
            BuildingComposer.mainDatabaseServer.abortTransaction();
        }
        // error
        return false;
    } // doIt
    // =====
    public boolean undoIt() {
        // if this command has not already been done then don't undoIt
        if (!m_Done) {
            return false;
        }
        try {
            BuildingComposer.mainDatabaseServer.startUpdateTransaction();
            // remove link
            m_curSite.unlinkContaine(m_curBuilding);
            BuildingComposer.mainDatabaseServer.commitTransaction();
            // success
            m_Done = false;
            return true;
        }
        catch (Exception e) {
            BuildingComposer.errorMessage(e);
            BuildingComposer.mainDatabaseServer.abortTransaction();
        }
        // error
        return false;
    } // undoIt
    // =====
    public Clfcbldg getBuilding() {
        return m_curBuilding;
    }
}
```

```

    }
}

```

A primary command can call one or more sub commands. Here is the interface that a primary command implements:

```

/** PrimaryCommand interface based on Command pattern. */
public interface PrimaryCommand extends Command {
    /**
     * Manages undo and redo functionality of commands.
     * Could be enhanced to include sequencing, logging, and scheduling functionality.
     */
    public final static CommandManager commandManager = new CommandManager();
    /**
     * PrimaryCommands should use this method to define the name of the
     * command followed by the name of the object it is performing on for
     * the purposes of the undo/redo GUI string.
     */
    public abstract String toString();
}

```

Here is an example of a primary command:

```

public class Cmd_AddBuilding implements PrimaryCommand {
    private Sub_AddBuilding mSub_addBuilding = null;
    private String m_label;
    // =====
    public Cmd_AddBuilding(ClfcSite site) {
        // construct subcommand
        mSub_addBuilding = new Sub_AddBuilding(site);
        m_label = new String("Add Building");
        // register with CommandManager
        try {
            commandManager.invokeCommand(this);
        }
        catch (CommandException e) {
            BuildingComposer.errorMessage(e);
        }
    }
    // =====
    /** Method called by CommandManager to do and redo the command. Should not be called directly.*/
    public boolean doIt() {
        return mSub_addBuilding.doIt();
    }
    // doIt
    // =====
    /** Method called by CommandManager for undo. Should not be called directly.*/
    public boolean undoIt() {
        return mSub_addBuilding.undoIt();
    }
    // undoIt
    // =====
    public String toString() {
        return m_label;
    }
}

```

In the *Building Composer* core library, the CommandManager class maintains the undo/redo lists and calling the doIt method for a primary command. When constructed a primary command notifies the static CommandManager instance of its creation. The CommandManager then calls the doIt method of the primary command. Here is some code from the CommandManager class:

```

public class CommandManager {
    // =====
    /**
     * Invoke a command and add it to the history, The command will be added
     * to the command history if doIt returns true.

```

```

*/
// =====
public void invokeCommand(Command command) throws CommandException {

    // call the commands dolt method
    try {
        if (command.dolt()) {
            addToDoneList(command);
        }
    }
    catch (Exception e) {
        //intercept all exceptions so that we can end command nicely
        BuildingComposer.errorMessage(e);
    }
} // invokeCommand
// =====
/**
 * Undo the most recent command in the command history.
 */
// =====
protected void undo() {
    if (m_doneList.isEmpty() == false) { // If there are commands in the m_doneList
        Command undoCommand;
        // get and remove first element in the m_doneList
        undoCommand = (Command)m_doneList.firstElement();
        m_doneList.removeElementAt(0);
        // call the commands undolt method
        try {
            undoCommand.undolt();
        }
        catch (Exception e) {
            //intercept all exceptions so that we can end command nicely
            BuildingComposer.errorMessage(e);
        }
        // add command that was just undone into redo list
        m_redoList.insertElementAt(undoCommand,0);
    }
} // undo
// =====
/**
 * Redo the most recently undone command.
 */
// =====
protected void redo() {
    if (m_redoList.isEmpty() == false) { // If the redo list is not empty
        Command redoCommand;
        // get and remove first element in m_redoList list
        redoCommand = (Command)m_redoList.firstElement();
        m_redoList.removeElementAt(0);
        // call the commands dolt method
        try {
            redoCommand.dolt();
        }
        catch (Exception e) {
            //intercept all exceptions so that we can end command nicely
            BuildingComposer.errorMessage(e);
        }
        // add command that was just undone into m_doneList
        m_doneList.insertElementAt(redoCommand,0);
    }
} // redo
}

```

With the separation of commands into primary and sub, an extension application can develop their own commands that implement the PrimaryCommand interface that will automatically follow the command design pattern. These custom commands may also use the sub commands that have been developed within the *Building Composer* core library.

5 Criteria Composer Application

The Criteria Composer application builds on top of the *Building Composer* core library to provide a complete graphical user interface to input, display, and manipulate data within the database. The main class for the Criteria Composer application is `Frame1`. This class creates a `JFrame` that contains an instance of the `ProjectCompositionPanel` and `DesignCriteriaPanel` class. This class also implements the `Int_MainFrame` interface. This is the interface that the *Building Composer* core library uses to interact with the main frame (or application). The *Building Composer* core library will “ask” the main frame at specific times for information or to inquire about appropriate action. Here is the `Int_MainFrame` interface:

```
public interface Int_MainFrame {
    /** Routine to return JFrame. */
    public JFrame getJFrame();
    /** Routine to return ProjectTreeUtils object to modify tree. */
    public ProjectTreeUtils getTreeUtils();
    /** Called after the database has been opened. Parameter 'title' should be added to frame title.
     The database will be open! */
    public void createOrOpenDB(String title);
    /** Called before the database is closed. Write out any database level information.
     The database will be open! */
    public void preCloseDB();
    /** Called after the database is closed.
     The database will NOT be open! */
    public void postCloseDB();
    /** Called when File->Exit is selected. This routine gives the application an opportunity to disallow the shutdown.
     The database will be open! */
    public boolean shutDownOK();
    /** Called when File->Exit is selected. Write out any session level information.
     Upon return BuildingComposer will call System.exit!
     The database will NOT be open! */
    public void shutDown();
}
```

6 Future Work

Though a good and fairly robust object model, several items of concern or places for improvement should be mentioned. Some these items (listed in no particular order) are:

1. *The database does no “garbage collection.”* Though items are added to the database automatically through transitive persistence whenever a transaction is closed, there is no automatic deletion, or removal, of an object. Once added to the database, an object will remain in it until specifically removed, even though no other object in the database may reference it (or if may no longer be used). Methods with the ObjectStore PSE database do allow for searching and removal of non-referenced objects within a database. However, in some cases, a disused object may still be of some use. The best approach to solve this may be the implementation of a “delete” method within all classes in the object model. This method would then be responsible for removing itself and variables from the database. One item of note here is that in the current implementation of the object model, the classes do not know of the database itself. The fundamental question is “Should an object know about how it is stored?” and “What if that storage approach changes?”
2. *Event messages generated from within the object model.* There are several places within the *Building Composer* core library where listeners can be added/removed to receive event messages when an object is manipulated. A better approach may be to have the object model elements themselves actually generate the event messages. This way the elements can send event messages whenever an element is created, manipulated, or deleted.
3. *The current object model is based on the IFC 1.5 version.* Changes to the object model would render existing projects useless. Several different approaches could be used within this specific case, although some applications and their datasets will doubtlessly experience problems. One option would be to implement the next version of the object model in its own package structure, and allowing the possibility of loading two versions of the dataset at the same time. A utility application could then open the older version and map it into the newer version. A possibly better approach would be to write our own “output” file, in XML format for example, that both versions would be able to read and write. The user would open the old versioned project, write a common file, open a blank new versioned project, and then read a common file.

4. *Less frequent database transactions.* The *Building Composer* core library takes care of most database transactions needed within it. One of the problems however is that in critical, time-sensitive applications or sections of code these transactions can reduce overall system performance. This was a concern in the Layout Composer application. During sections of code that deal with the graphics system and its redrawing of a graphical element, the system was too slow to present a “smooth” redraw of the element. To alleviate this, the Layout Composer application maintains an internal count of the number of commands performed. Using this count the Layout Composer application starts an update transaction at count one and then commits the transaction at count 10. This change greatly improved the redraw rate of an element. The drawback of this solution is that, if there is an application failure, any changes done since the last transaction commit are lost.

Through the use of the object model and *Building Composer* core library it is possible to write many applications that can open, modify, and close a facility project. The object model was written to be an open model that can be used directly through methods defined by the IFC object model or through custom controller classes developed. The *Building Composer* core library can easily be used as a basis for a final application and can be hooked into to receive event messages when certain actions are performed. This report has discussed many of the key concepts and classes within these pieces.

References

IAI IFC Documentation, Version 1.5. <http://iaiweb.lbl.gov/>

StepTools Documentation, Version 7.0. StepTools, Inc; 1223 Peoples Avenue; Troy, New York; 12180; (518) 276-2848. <http://www.steptools.com>

ObjectStore PSE Pro Documentation, Version 6.0. eXcelon Corp.; 25 Mall Road; Burlington, MA; 01803; 1-800-424-9797. <http://www.exceloncorp.com/> or <http://www.objectdesign.com/>

Sun Java Development Kit Documentation, Version 1.2. Sun Microsystems, Inc.; 901 San Antonio Road; Palo Alto, CA; 94303. <http://www.javasoft.com/>

Appendix A: IFC 1.50 EXPRESS File

```

SCHEMA Ifc150Final;
-- IfcArchitecture - Type Definitions
TYPE IfcSpaceProgramTypeEnum = ENUMERATION OF (
    SpaceStandard
    ,SpaceUnique);
END_TYPE;
-- IfcArchitecture - Entity Definitions
ENTITY IfcSpaceProgramGroup
    SUBTYPE OF (IfcGroup);
    RequiredGroupArea : OPTIONAL IfcAreaMeasure;
    GroupRole : STRING;
    GroupAssignment : IfcActorSelect;
    WHERE
        WR1: SIZEOF(QUERY(Temp <* SELF\IfcGroup.GroupedBy.RelatedObjects |
            'IFC150FINAL.IFCSPACEPROGRAM' IN TYPEOF(Temp))) >= 1;
END_ENTITY;
ENTITY IfcRelAdjacencyReq
    SUBTYPE OF (IfcRelationship1to1);
    -- SELF\IfcRelationship1to1.RelatingObject : IfcSpaceProgram;
    -- SELF\IfcRelationship1to1.RelatedObject : IfcSpaceProgram;
    RequiredAdjacency : INTEGER;
END_ENTITY;
ENTITY IfcSpaceProgram
    SUBTYPE OF (IfcControl);
    SpaceName : STRING;
    GenericType : IfcSpaceProgramTypeEnum;
    ProgramForSpaces : SET [1:?] OF IfcSpace;
    INVERSE
        HasAdjacencyReqsTo : SET[0:?] OF IfcRelAdjacencyReq FOR RelatingObject;
        HasAdjacencyReqFrom : SET[0:?] OF IfcRelAdjacencyReq FOR RelatedObject;
END_ENTITY;
-- IfcDocumentExtension - TYPE Definition
TYPE IfcCostElementOrGroupSelect = SELECT (
    IfcCostElementGroup
    ,IfcCostElement );
END_TYPE;
-- IfcDocumentExtension - ENTITY Definition
ENTITY IfcCostSchedule
    SUBTYPE OF (IfcDocument);
    ScheduleTitle : STRING;
    SubmittedBy : OPTIONAL IfcActorSelect;
    ApprovedBy : OPTIONAL IfcPerson;
    PreparedBy : OPTIONAL IfcPerson;
    SubmittedOn : OPTIONAL IfcDateTimeSelect;
    Cost : OPTIONAL IfcCost;
    HasCostElementGroup : LIST [1:?] OF IfcCostElementGroup;
END_ENTITY;
ENTITY IfcCostElementGroup
    SUBTYPE OF (IfcGroup);
    GroupTitle : OPTIONAL STRING;
    GroupCos : OPTIONAL IfcCost;
    PreparedOn : OPTIONAL IfcDateTimeSelect;
    GroupId : OPTIONAL STRING;
    INVERSE
        PartOfCostSchedule : IfcCostSchedule FOR HasCostElementGroup;
    WHERE
        WR1: SIZEOF(QUERY(TEMP <* SELF\IfcGroup.GroupedBy.RelatedObjects |
            'IFC150FINAL.IFCCOSTELEMENT' IN TYPEOF(Temp))) >= 1;
END_ENTITY;

```

```

ENTITY IfcCostElement
SUBTYPE OF (IfcControl);
  Description : OPTIONAL STRING;
  ElementCost : OPTIONAL IfcCost;
  ExtensionCost : OPTIONAL IfcCost;
WHERE
  WR1: SIZEOF(QUERY(TEMP <* SELF\IfcObject.PartOfGroups |
    'IFC150FINAL.IFCCOSTELEMENTGROUP' IN TYPEOF(Temp.RelatingObject))) = 1;
END_ENTITY;
ENTITY IfcRelCostScheduleElement
SUBTYPE OF (IfcRelationship1toN);
-- SELF\IfcRelationship1toN.RelatedObjects : LIST [1:?] OF IfcProduct;
  Quantity : NUMBER;
WHERE
  ('IFC150FINAL.IFCCOSTELEMENTGROUP' IN TYPEOF (SELF\IfcRelationship1toN.RelatingObject))
  OR
  ('IFC150FINAL.IFCCOSTELEMENT' IN TYPEOF (SELF\IfcRelationship1toN.RelatingObject));
END_ENTITY;
-- IfcFacilitiesMgmt - Cross schema references
-- IfcFacilitiesMgmt - Type Definitions
TYPE IfcFurnitureTypeEnum = ENUMERATION OF (
  Table
  ,Chair
  ,Desk
  ,FileCabinet);
END_TYPE;
-- IfcFacilitiesMgmt - Entity Definitions
ENTITY IfcFurniture
SUBTYPE OF (IfcBuildingElement);
  GenericType : IfcFurnitureTypeEnum;
  AssignedTo : OPTIONAL IfcActorSelect;
END_ENTITY;
-- IfcGeometryResource - Type Definitions for Explicit
TYPE IfcDimensionCount = INTEGER;
WHERE
  WR1: { 0 < SELF <= 3 };
END_TYPE;
TYPE IfcTransitionCode = ENUMERATION OF (
  Discontinuous
  ,Continuous
  ,ContSameGradient
  ,ContSameGradientSameCurvature);
END_TYPE;
TYPE IfcTrimmingPreference = ENUMERATION OF (
  Cartesian
  ,Parameter
  ,Unspecified);
END_TYPE;
TYPE IfcAxis2Placement = SELECT (
  IfcAxis2Placement2D
  ,IfcAxis2Placement3D);
END_TYPE;
TYPE IfcTrimmingSelect = SELECT (
  IfcCartesianPoint
  ,IfcParameterValue);
END_TYPE;
TYPE IfcVectorOrDirection = SELECT (
  IfcVector
  ,IfcDirection);
END_TYPE;
-- IfcGeometryResource - Type Definitions for Attribute Driven
TYPE IfcProfileTypeEnum = ENUMERATION OF (
  Curve
  ,Area);
END_TYPE;
-- IfcGeometryResource - Entity Definitions for Explicit
ENTITY IfcAxis1Placement
SUBTYPE OF (IfcPlacement);

```

```

    Axis : OPTIONAL IfcDirection;
DERIVE
  Z : IfcDirection
    := NVL (
      IfcNormalise(Axis)
      ,IfcGeometricRepresentationItem() || IfcDirection([0.0,0.0,1.0]));
  Dim : IfcDimensionCount
    := 3;
WHERE
  WR1 : (NOT (EXISTS (Axis))) OR (Axis.Dim = 3);
END_ENTITY;
ENTITY IfcAxis2Placement2D
SUBTYPE OF (IfcPlacement);
  RefDirection : OPTIONAL IfcDirection;
DERIVE
  P : LIST [2:2] OF IfcDirection
    := IfcBuild2Axes(RefDirection);
  Dim : IfcDimensionCount
    := 2;
WHERE
  WR1: (NOT (EXISTS (RefDirection))) OR (RefDirection.Dim = 2);
END_ENTITY;
ENTITY IfcAxis2Placement3D
SUBTYPE OF (IfcPlacement);
  Axis : OPTIONAL IfcDirection;
  RefDirection : OPTIONAL IfcDirection;
DERIVE
  P : LIST [3:3] OF IfcDirection
    := IfcBuildAxes(Axis, RefDirection);
  Dim : IfcDimensionCount
    := 3;
WHERE
  WR1: SELF.IfPlacement.Location.Dim = 3;
  WR2: (NOT (EXISTS (Axis))) OR (Axis.Dim = 3);
  WR3: (NOT (EXISTS (RefDirection))) OR (RefDirection.Dim = 3);
  WR4: (NOT (EXISTS (Axis))) OR (NOT (EXISTS (RefDirection))) OR
    (IfcCrossProduct(Axis,RefDirection).Magnitude > 0.0);
  WR5: NOT ((EXISTS (Axis)) XOR (EXISTS (RefDirection)));
END_ENTITY;
ENTITY IfcBoundedCurve
ABSTRACT SUPERTYPE OF (ONEOF(
  IfcPolyline
  ,IfcTrimmedCurve
  ,IfcCompositeCurve))
SUBTYPE OF (IfcCurve);
END_ENTITY;
ENTITY IfcBoundingBox
SUBTYPE OF (
  IfcGeometricRepresentationItem );
  Position : IfcAxis2Placement3D;
  XDim : IfcPositiveLengthMeasure;
  YDim : IfcPositiveLengthMeasure;
  ZDim : IfcPositiveLengthMeasure;
DERIVE
  Dim : IfcDimensionCount
    := 3;
END_ENTITY;
ENTITY IfcCartesianPoint
SUBTYPE OF (IfcPoint);
  Coordinates : LIST [1:3] OF IfcLengthMeasure;
DERIVE
  Dim : IfcDimensionCount
    := HINDEX(Coordinates);
WHERE
  WR1 : HINDEX(Coordinates) >= 2;
END_ENTITY;
ENTITY IfcCircle
SUBTYPE OF (IfcConic);

```

```

    Radius : IfcPositiveLengthMeasure;
END_ENTITY;
ENTITY IfcClosedShell
  SUBTYPE OF (IfcConnectedFaceSet);
END_ENTITY;
ENTITY IfcCompositeCurve
  SUPERTYPE OF (ONEOF (Ifc2DCompositeCurve))
  SUBTYPE OF (IfcBoundedCurve);
  Segments : LIST [1:?] OF IfcCompositeCurveSegment;
  SelfIntersect : LOGICAL;
DERIVE
  NSegments : INTEGER
    := SIZEOF(Segments);
  ClosedCurve : LOGICAL
    := Segments[NSegments].Transition <> Discontinuous;
  Dim : IfcDimensionCount
    := Segments[1].Dim;
WHERE
  WR1: ((NOT ClosedCurve) AND (SIZEOF(QUERY(Temp <* Segments |
    Temp.Transition = Discontinuous)) = 1)) OR
    ((ClosedCurve) AND (SIZEOF(QUERY(Temp <* Segments |
    Temp.Transition = Discontinuous)) = 0));
  WR2: SIZEOF( QUERY( Temp <* Segments | Temp.Dim <>
    Segments[1].Dim)) = 0;
END_ENTITY;
ENTITY Ifc2DCompositeCurve
  SUBTYPE OF (IfcCompositeCurve);
WHERE
  WR1: SELF.IfCompositeCurve.ClosedCurve;
  WR2: Dim = 2;
END_ENTITY;
ENTITY IfcCompositeCurveSegment
  SUBTYPE OF (IfcGeometricRepresentationItem);
  Transition : IfcTransitionCode;
  SameSense : BOOLEAN;
  ParentCurve : IfcCurve;
DERIVE
  Dim : IfcDimensionCount
    := ParentCurve.Dim;
INVERSE
  UsingCurves : BAG[1:?] OF IfcCompositeCurve FOR Segments;
WHERE
  WR1 : ('IFC150FINAL.IFCBOUNDED_CURVE' IN TYPEOF(ParentCurve));
END_ENTITY;
ENTITY IfcConic
  ABSTRACT SUPERTYPE OF (ONEOF(
    IfcCircle
    ,IfcEllipse))
  SUBTYPE OF (IfcCurve);
  Position : IfcAxis2Placement;
DERIVE
  Dim : IfcDimensionCount
    := SELF.Position.Dim;
END_ENTITY;
ENTITY IfcConnectedFaceSet
  SUPERTYPE OF (ONEOF(IfcClosedShell))
  SUBTYPE OF (IfcTopologicalRepresentationItem);
  CfsFaces : SET [1:?] OF IfcFace;
END_ENTITY;
ENTITY IfcCurve
  ABSTRACT SUPERTYPE OF (ONEOF(
    IfcLine
    ,IfcConic
    ,IfcBoundedCurve))
  SUBTYPE OF (IfcGeometricRepresentationItem);
END_ENTITY;
ENTITY IfcCurveBoundedPlane
  SUBTYPE OF (IfcSurface);

```

```

    BasisSurface : IfcPlane;
    OuterBoundary : Ifc2DCompositeCurve;
    InnerBoundaries : SET [0:?] OF Ifc2DCompositeCurve;
DERIVE
    Dim : IfcDimensionCount
    := SELF.BasisSurface.Dim;
END_ENTITY;
ENTITY IfcDirection
SUBTYPE OF (IfcGeometricRepresentationItem);
    DirectionRatios : LIST [2:3] OF REAL;
DERIVE
    Dim : IfcDimensionCount
    := HINDEX(DirectionRatios);
WHERE
    WR1: SIZEOF(QUERY(Temp <* DirectionRatios | Temp <> 0.0)) > 0;
END_ENTITY;
ENTITY IfcElementarySurface
ABSTRACT SUPERTYPE OF (ONEOF(IfcPlane))
SUBTYPE OF (IfcSurface);
    Position : IfcAxis2Placement3D;
DERIVE
    Dim : IfcDimensionCount
    := Position.Dim;
END_ENTITY;
ENTITY IfcEllipse
SUBTYPE OF (IfcConic);
    SemiAxis1 : IfcPositiveLengthMeasure;
    SemiAxis2 : IfcPositiveLengthMeasure;
END_ENTITY;
ENTITY IfcExtrudedAreaSolid
SUBTYPE OF (IfcSweptAreaSolid);
    ExtrudedDirection : IfcDirection;
    Depth : IfcPositiveLengthMeasure;
WHERE
    WR1: IfcDotProduct(
        (SELF.IfcsweptAreaSolid.SweptArea.BasisSurface\
        IfcPlane.Position.P[3]), ExtrudedDirection) <> 0.0;
END_ENTITY;
ENTITY IfcFace
SUBTYPE OF (IfcTopologicalRepresentationItem);
    Bounds : SET [1:?] OF IfcFaceBound;
WHERE
    WR2: SIZEOF(QUERY(temp <* Bounds | 'IFC150FINAL.IFCFACEOUTERBOUND'
        IN TYPEOF(temp))) <= 1;
END_ENTITY;
ENTITY IfcFaceBound
SUPERTYPE OF (ONEOF(
    IfcFaceOuterBound))
SUBTYPE OF (IfcTopologicalRepresentationItem);
    Bound : IfcPolyLoop;
    Orientation : BOOLEAN;
END_ENTITY;
ENTITY IfcFaceOuterBound
SUBTYPE OF (IfcFaceBound);
END_ENTITY;
ENTITY IfcFacetedBrep
SUBTYPE OF (IfcManifoldSolidBrep);
END_ENTITY;
ENTITY IfcFacetedBrepWithVoids
SUBTYPE OF (IfcManifoldSolidBrep);
    Voids : SET [1:?] OF IfcClosedShell;
END_ENTITY;
ENTITY IfcGeometricRepresentationItem
ABSTRACT SUPERTYPE OF (ONEOF(
    IfcBoundingBox
    ,IfcPoint
    ,IfcCurve
    ,IfcDirection

```

```

,IfcPolyLoop
,IfcPlacement
,IfcSurface
,IfcCompositeCurveSegment
,IfcHalfSpaceSolid
,IfcSolidModel
,IfcVector ));
END_ENTITY;
ENTITY IfcHalfSpaceSolid
SUBTYPE OF (IfcGeometricRepresentationItem);
  BaseSurface : IfcSurface;
  AgreementFlag : BOOLEAN;
DERIVE
  Dim : IfcDimensionCount
  := 3;
END_ENTITY;
ENTITY IfcLine
SUBTYPE OF (IfcCurve);
  Pnt : IfcCartesianPoint;
  Dir : IfcVector;
DERIVE
  Dim : IfcDimensionCount
  := Pnt.Dim;
WHERE
  WR1 : Dir.Dim = Pnt.Dim;
END_ENTITY;
ENTITY IfcManifoldSolidBrep
ABSTRACT SUPERTYPE OF (ONEOF (
  IfcFacetedBrep
  ,IfcFacetedBrepWithVoids))
SUBTYPE OF (IfcSolidModel);
  Outer : IfcClosedShell;
END_ENTITY;
ENTITY IfcPlacement
ABSTRACT SUPERTYPE OF (ONEOF(
  IfcAxis1Placement
  ,IfcAxis2Placement2D
  ,IfcAxis2Placement3D))
SUBTYPE OF (IfcGeometricRepresentationItem);
  Location : IfcCartesianPoint;
END_ENTITY;
ENTITY IfcPlane
SUBTYPE OF (IfcElementarySurface);
END_ENTITY;
ENTITY IfcPoint
ABSTRACT SUPERTYPE OF (ONEOF (IfcCartesianPoint))
SUBTYPE OF (IfcGeometricRepresentationItem);
END_ENTITY;
ENTITY IfcPolyline
SUBTYPE OF (IfcBoundedCurve);
  Points : LIST [2:?] OF IfcCartesianPoint;
DERIVE
  Dim : IfcDimensionCount
  := Points[1].Dim;
WHERE
  WR1: SIZEOF(QUERY(Temp <* Points | Temp.Dim <> Points[1].Dim))
  = 0;
END_ENTITY;
ENTITY IfcPolyLoop
SUBTYPE OF (IfcGeometricRepresentationItem);
  Polygon : LIST [3:?] OF UNIQUE IfcCartesianPoint;
DERIVE
  Dim : IfcDimensionCount
  := Polygon[1].Dim;
WHERE
  WR1: SIZEOF(QUERY(Temp <* Polygon | Temp.Dim <> Polygon[1].Dim))
  = 0;
END_ENTITY;

```

```

ENTITY IfcRevolvedAreaSolid
SUBTYPE OF (IfcSweptAreaSolid);
  Axis : IfcAxis1Placement;
  Angle : IfcPlaneAngleMeasure;
DERIVE
  AxisLine : IfcLine
    := IfcGeometricRepresentationItem() || IfcLine
      (Axis.Location
        ,IfcGeometricRepresentationItem() || IfcVector(Axis.Z,1.0));
END_ENTITY;
ENTITY IfcSolidModel
ABSTRACT SUPERTYPE OF (ONEOF(
  ,IfcManifoldSolidBrep
  ,IfcSweptAreaSolid
  ,IfcAttDrivenExtrudedSolid
  ,IfcAttDrivenRevolvedSolid))
SUBTYPE OF (IfcGeometricRepresentationItem);
DERIVE
  Dim : IfcDimensionCount
    := 3;
END_ENTITY;
ENTITY IfcSurface
ABSTRACT SUPERTYPE OF (ONEOF(
  ,IfcElementarySurface
  ,IfcCurveBoundedPlane))
SUBTYPE OF (IfcGeometricRepresentationItem);
END_ENTITY;
ENTITY IfcSweptAreaSolid
ABSTRACT SUPERTYPE OF (ONEOF(
  ,IfcExtrudedAreaSolid
  ,IfcRevolvedAreaSolid))
SUBTYPE OF (IfcSolidModel);
  SweptArea : IfcCurveBoundedPlane;
WHERE
  WR1: 'IFC150FINAL.IFCPLANE' IN
    TYPEOF(SweptArea.BasisSurface);
END_ENTITY;
ENTITY IfcTopologicalRepresentationItem
ABSTRACT SUPERTYPE OF (ONEOF(
  ,IfcConnectedFaceSet
  ,IfcFace
  ,IfcFaceBound));
END_ENTITY;
ENTITY IfcTrimmedCurve
SUBTYPE OF (IfcBoundedCurve);
  BasisCurve : IfcCurve;
  Trim1 : SET[1:2] OF IfcTrimmingSelect;
  Trim2 : SET[1:2] OF IfcTrimmingSelect;
  SenseAgreement : BOOLEAN;
  MasterRepresentation : IfcTrimmingPreference;
DERIVE
  Dim : IfcDimensionCount
    := SELF.BasisCurve.Dim;
WHERE
  WR1: (HIINDEX(Trim1) = 1) XOR (TYPEOF(Trim1[1]) <> TYPEOF(Trim1[2]));
  WR2: (HIINDEX(Trim2) = 1) XOR (TYPEOF(Trim2[1]) <> TYPEOF(Trim2[2]));
  WR3: NOT('IFC150FINAL.IFCBOUNDEDCURVE' IN TYPEOF(SELF.BasisCurve));
END_ENTITY;
ENTITY IfcVector
SUBTYPE OF (IfcGeometricRepresentationItem);
  Orientation : IfcDirection;
  Magnitude : IfcLengthMeasure;
DERIVE
  Dim : IfcDimensionCount
    := Orientation.Dim;
WHERE
  WR1: Magnitude >= 0.0;
END_ENTITY;

```



```

-- IfcGeometryResource - Entity Definitions for AttDriven
ENTITY IfcAttDrivenExtrudedSolid
SUPERTYPE OF (ONEOF (
  IfcAttDrivenClippedExtrudedSolid))
SUBTYPE OF (IfcSolidModel);
  Segments      : LIST [1:?] OF IfcAttDrivenExtrudedSegment;
DERIVE
  Path          : IfcPolyline
                := IfcExtrusionPath(SELF);
WHERE
  WR1: SIZEOF(QUERY( Temp <* Segments |
    Temp.Position.Axis <> Segments[1].Position.Axis))
    = 0;
END_ENTITY;
ENTITY IfcAttDrivenClippedExtrudedSolid
SUBTYPE OF (IfcAttDrivenExtrudedSolid);
  ClippingHalfSpaces : LIST [1:2] OF IfcHalfSpaceSolid;
END_ENTITY;
ENTITY IfcAttDrivenExtrudedSegment
SUPERTYPE OF (ONEOF(IfcAttDrivenMorphedExtrudedSegment,IfcAttDrivenTaperedExtrudedSegment))
SUBTYPE OF (IfcExtrudedAreaSolid);
  Position      : IfcAxis2Placement3D;
  ProfileDef     : IfcAttDrivenProfileDef;
DERIVE
  SELF\IfcSweptAreaSolid.SweptArea : IfcCurveBoundedPlane
    := IfcProfileIntoArea(ProfileDef);
  SELF\IfcExtrudedAreaSolid.ExtrudedDirection : IfcDirection
    := IfcGeometricRepresentationItem() || IfcDirection([0.0,0.0,1.0]);
INVERSE
  PartOfSolid : IfcAttDrivenExtrudedSolid FOR Segments;
END_ENTITY;
ENTITY IfcAttDrivenTaperedExtrudedSegment
SUBTYPE OF (IfcAttDrivenExtrudedSegment);
  TaperingFactor : IfcPositiveRatioMeasure;
END_ENTITY;
ENTITY IfcAttDrivenMorphedExtrudedSegment
SUBTYPE OF (IfcAttDrivenExtrudedSegment);
  EndProfileDef : IfcAttDrivenProfileDef;
DERIVE
  EndSweptArea : IfcCurveBoundedPlane
    := IfcProfileIntoArea(EndProfileDef);
WHERE
  WR1: TYPEOF(SELF\IfcAttDrivenExtrudedSegment.ProfileDef) = TYPEOF(EndProfileDef);
  WR2: NOT('IFC150FINAL.IFCARBITRARYPROFILEDEF'
    IN TYPEOF(SELF\IfcAttDrivenRevolvedSegment.ProfileDef));
  WR3: SELF\IfcAttDrivenExtrudedSegment.ProfileDef.Position.P[1] = EndProfileDef.Position.P[1];
END_ENTITY;
ENTITY IfcAttDrivenRevolvedSolid
SUPERTYPE OF (ONEOF (
  IfcAttDrivenClippedRevolvedSolid))
SUBTYPE OF (IfcSolidModel);
  Segments      : LIST [1:?] OF IfcAttDrivenRevolvedSegment;
DERIVE
  Path          : IfcTrimmedCurve
                := IfcRevolutionPath(SELF);
WHERE
  WR1: SIZEOF(QUERY( Temp <* Segments |
    Temp.Position :<> Segments[1].Position))
    = 0;
  WR2: SIZEOF(QUERY( Temp <* Segments |
    Temp\IfcRevolvedAreaSolid.Axis <> Segments[1]\IfcRevolvedAreaSolid.Axis))
    = 0;
END_ENTITY;
ENTITY IfcAttDrivenClippedRevolvedSolid
SUBTYPE OF (IfcAttDrivenRevolvedSolid);
  ClippingHalfSpaces : LIST [1:2] OF IfcHalfSpaceSolid;
END_ENTITY;
ENTITY IfcAttDrivenRevolvedSegment

```

```

SUPERTYPE OF (ONEOF(IfcAttDrivenMorphedRevolvedSegment,IfcAttDrivenTaperedRevolvedSegment))
SUBTYPE OF (IfcRevolvedAreaSolid);
  Position      : IfcAxis2Placement3D;
  StartAngle    : IfcPlaneAngleMeasure;
  ProfileDef    : IfcAttDrivenProfileDef;
DERIVE
  SELF\IfcSweptAreaSolid.SweptArea : IfcCurveBoundedPlane
    := IfcProfileIntoArea(ProfileDef);
INVERSE
  PartOfSolid : IfcAttDrivenRevolvedSolid FOR Segments;
WHERE
  WR1: SELF\IfcRevolvedAreaSolid.Axis.Location.Coordinates[3] = 0;
END_ENTITY;
ENTITY IfcAttDrivenTaperedRevolvedSegment
SUBTYPE OF (IfcAttDrivenRevolvedSegment);
  TaperingFactor : IfcPositiveRatioMeasure;
END_ENTITY;
ENTITY IfcAttDrivenMorphedRevolvedSegment
SUBTYPE OF (IfcAttDrivenRevolvedSegment);
  EndProfileDef : IfcAttDrivenProfileDef;
DERIVE
  EndSweptArea : IfcCurveBoundedPlane
    := IfcProfileIntoArea(EndProfileDef);
WHERE
  WR1: TYPEOF(SELF\IfcAttDrivenRevolvedSegment.ProfileDef) = TYPEOF(EndProfileDef);
  WR2: NOT('IFC150FINAL.IFCARBITRARYPROFILEDEF'
    IN TYPEOF(SELF\IfcAttDrivenRevolvedSegment.ProfileDef));
  WR3: SELF\IfcAttDrivenRevolvedSegment.ProfileDef.Position.P[1] = EndProfileDef.Position.P[1];
END_ENTITY;
ENTITY IfcAttDrivenProfileDef
ABSTRACT SUPERTYPE OF (ONEOF(
  ,IfcRectangleProfileDef
  ,IfcCircleProfileDef
  ,IfcTrapeziumProfileDef
  ,IfcArbitraryProfileDef));
  Position      : IfcAxis2Placement2D;
  ProfileType   : IfcProfileTypeEnum;
DERIVE
  PositionToOrigin : LIST[2:2] OF IfcLengthMeasure
    := Position.Location.Coordinates;
  AngleInOrigin : IfcPlaneAngleMeasure
    := IfcComputeAngleFromAxis(Position.P[1]);
END_ENTITY;
ENTITY IfcArbitraryProfileDef
SUBTYPE OF (IfcAttDrivenProfileDef);
  CurveForSurface : IfcBoundedCurve;
WHERE
  WR1: (('IFC150FINAL.IFCPOLYLINE' IN
    TYPEOF(CurveForSurface)) AND (CurveForSurface.Dim = 2))
    OR
    (('IFC150FINAL.IFCTRIMMEDCURVE' IN
    TYPEOF(CurveForSurface)) AND (CurveForSurface.Dim = 2))
    OR
    (('IFC150FINAL.IFCCOMPOSITECURVE' IN
    TYPEOF(CurveForSurface)) AND (CurveForSurface.Dim = 2));
END_ENTITY;
ENTITY IfcCircleProfileDef
SUBTYPE OF (IfcAttDrivenProfileDef);
  Radius : IfcPositiveLengthMeasure;
DERIVE
  CurveForSurface : IfcTrimmedCurve
    := IfcCircleProfileIntoCurve(SELF);
END_ENTITY;
ENTITY IfcRectangleProfileDef
SUBTYPE OF (IfcAttDrivenProfileDef);
  XDim : IfcPositiveLengthMeasure;
  YDim : IfcPositiveLengthMeasure;
DERIVE

```

```

    CurveForSurface : IfcPolyline
        := IfcRectangleProfileIntoCurve(SELF);
END_ENTITY;
ENTITY IfcTrapeziumProfileDef
SUBTYPE OF (IfcAttDrivenProfileDef);
    BottomXDim : IfcPositiveLengthMeasure;
    TopXDim : IfcPositiveLengthMeasure;
    YDim : IfcPositiveLengthMeasure;
    TopXOffset : IfcLengthMeasure;
DERIVE
    CurveForSurface : IfcPolyline
        := IfcTrapeziumProfileIntoCurve(SELF);
END_ENTITY;
-- IfcGeometryResource - Function Definition for Explicit
FUNCTION IfcBuildAxes (
    Axis, RefDirection : IfcDirection
    : LIST [3:3] OF IfcDirection;
LOCAL
    U : LIST [3:3] OF IfcDirection
        := [IfcGeometricRepresentationItem() || IfcDirection([1.0,0.0,0.0])
        ,IfcGeometricRepresentationItem() || IfcDirection([0.0,1.0,0.0])
        ,IfcGeometricRepresentationItem() || IfcDirection([0.0,0.0,1.0])];
END_LOCAL;
    U[3] := NVL(IfcNormalise(Axis)
        ,IfcGeometricRepresentationItem() || IfcDirection([0.0,0.0,1.0]));
    U[1] := IfcFirstProjAxis(U[3], RefDirection);
    U[2] := IfcNormalise(IfcCrossProduct(U[3],U[1])).Orientation;
    RETURN(U);
END_FUNCTION;
FUNCTION IfcBuild2Axes (
    RefDirection : IfcDirection
    : LIST [2:2] OF IfcDirection;
LOCAL
    U : LIST[2:2] OF IfcDirection
        := [IfcGeometricRepresentationItem() || IfcDirection([1.0,0.0])
        ,IfcGeometricRepresentationItem() || IfcDirection([0.0,1.0])];
END_LOCAL;
    U[1] := NVL(IfcNormalise(RefDirection)
        ,IfcGeometricRepresentationItem() || IfcDirection([1.0,0.0]));
    U[2] := IfcOrthogonalComplement(U[1]);
    RETURN(U);
END_FUNCTION;
FUNCTION IfcCrossProduct (
    Arg1, Arg2 : IfcDirection
    : IfcVector;
LOCAL
    Mag : REAL := 0.0;
    V1,V2 : LIST[3:3] OF REAL := [0.0:3];
    Res : IfcDirection
        := IfcGeometricRepresentationItem() || IfcDirection([1.0,0.0,0.0]);
    Result : IfcVector
        := IfcGeometricRepresentationItem() || IfcVector (
            IfcGeometricRepresentationItem() || IfcDirection([1.0,0.0,0.0])
            ,1.0);
END_LOCAL;
    IF ( NOT EXISTS (Arg1) OR (Arg1.Dim = 2)) OR
        ( NOT EXISTS (Arg2) OR (Arg2.Dim = 2)) THEN
        RETURN(?);
    ELSE
    BEGIN
        V1 := IfcNormalise(Arg1).DirectionRatios;
        V2 := IfcNormalise(Arg2).DirectionRatios;
        Res.DirectionRatios[1] := (V1[2]*V2[3] - v1[3]*v2[2]);
        Res.DirectionRatios[2] := (V1[3]*V2[1] - v1[1]*v2[3]);
        Res.DirectionRatios[3] := (V1[1]*V2[2] - v1[2]*v2[1]);
        Mag := 0.0;
        REPEAT i := 1 TO 3;
            Mag := Mag + Res.DirectionRatios[i]*Res.DirectionRatios[i];

```

```

END_REPEAT;
IF (Mag > 0.0) THEN
  Result.Orientation := Res;
  Result.Magnitude := SQRT(Mag);
ELSE
  Result.Orientation := Arg1;
  Result.Magnitude := 0.0;
END_IF;
RETURN(Result);
END;
END_IF;
END_FUNCTION;
FUNCTION IfcDotProduct (Arg1, Arg2 : IfcDirection)
  : REAL;
LOCAL
  Scalar : REAL := 0.0;
  Ndim : INTEGER := 0;
  Vec1, Vec2 : IfcDirection
    := IfcGeometricRepresentationItem() || IfcDirection([1.0,0.0,0.0]);
END_LOCAL;
IF NOT EXISTS (Arg1) OR NOT EXISTS (Arg2) THEN
  Scalar := ?;
ELSE
  IF (Arg1.dim <> Arg2.dim) THEN
    Scalar := ?;
  ELSE
    BEGIN
      Vec1 := IfcNormalise(Arg1);
      Vec2 := IfcNormalise(Arg2);
      Ndim := Arg1.Dim;
      Scalar := 0.0;
      REPEAT i := 1 TO Ndim;
        Scalar := Scalar +
          Vec1.DirectionRatios[i] *
          Vec2.DirectionRatios[i];
      END_REPEAT;
    END;
  END_IF;
END_IF;
RETURN (Scalar);
END_FUNCTION;
FUNCTION IfcFirstProjAxis (ZAxis, Arg : IfcDirection)
  : IfcDirection;
LOCAL
  XAxis, V, Z : IfcDirection
    := IfcGeometricRepresentationItem() || IfcDirection([1.0,0.0,0.0]);
  XVec : IfcVector
    := IfcGeometricRepresentationItem() || IfcVector (
      IfcGeometricRepresentationItem() || IfcDirection([1.0,0.0,0.0])
      ,1.0);
END_LOCAL;
IF (NOT EXISTS(ZAxis)) OR (NOT EXISTS(Arg)) OR (Arg.Dim <> 3) THEN
  XAxis := ?;
ELSE
  ZAxis := IfcNormalise(ZAxis);
  IF NOT EXISTS(Arg) THEN
    IF (ZAxis <> IfcDirection([1.0,0.0,0.0])) THEN
      V := IfcGeometricRepresentationItem() || IfcDirection([1.0,0.0,0.0]);
    ELSE
      V := IfcGeometricRepresentationItem() || IfcDirection([0.0,1.0,0.0]);
    END_IF;
  ELSE
    IF ((IfcCrossProduct(Arg,Z).Magnitude) = 0.0) THEN
      RETURN (?);
    ELSE
      V := IfcNormalise(Arg);
    END_IF;
  END_IF;
END_IF;

```

```

XVec := IfcScalarTimesVector(IfcDotProduct(V,Z), ZAxis);
XAxis := IfcVectorDifference(V, XVec).Orientation;
XAxis := IfcNormalise(XAxis);
END_IF;
RETURN(XAxis);
END_FUNCTION;
FUNCTION IfcNormalise (
  Arg : IfcVectorOrDirection)
  : IfcVectorOrDirection;
LOCAL
  Ndim : INTEGER := 0;
  Mag : REAL := 0.0;
  V : IfcDirection
    := IfcGeometricRepresentationItem() || IfcDirection([1.0,0.0,0.0]);
  Vec : IfcVector
    := IfcGeometricRepresentationItem() || IfcVector (
      IfcGeometricRepresentationItem() || IfcDirection([1.0,0.0,0.0])
      ,1.0);
  Result : IfcVectorOrDirection
    := V;
END_LOCAL;
IF NOT EXISTS (Arg) THEN
  Result := ?;
ELSE
  Ndim := Arg.Dim;
  IF 'IFC150FINAL.IFCVECTOR' IN TYPEOF(Arg) THEN
    BEGIN
      Vec := Arg;
      V := Arg.Orientation;
      IF Arg.Magnitude = 0.0 THEN
        RETURN(?);
      ELSE
        Vec.Magnitude := 1.0;
      END_IF;
    END;
  ELSE
    V := Arg;
    END_IF;
    Mag := 0.0;
    REPEAT i := 1 TO Ndim;
      Mag := Mag + V.DirectionRatios[i] * V.DirectionRatios[i];
    END_REPEAT;
    IF Mag > 0.0 THEN
      Mag := SQRT(Mag);
      REPEAT i := 1 TO Ndim;
        V.DirectionRatios[i] := V.DirectionRatios[i]/Mag;
      END_REPEAT;
      IF 'IFC150FINAL.IFCVECTOR' IN TYPEOF(Arg) THEN
        Vec.Orientation := V;
        Result := Vec;
      ELSE
        Result := V;
      END_IF;
    ELSE
      RETURN(?);
    END_IF;
  END_IF;
  RETURN (Result);
END_FUNCTION;
FUNCTION IfcOrthogonalComplement (
  Vec : IfcDirection)
  : IfcDirection;
LOCAL
  Result : IfcDirection
    := IfcGeometricRepresentationItem() || IfcDirection([1.0,0.0,0.0]);
END_LOCAL;
IF (Vec.Dim <> 2) OR NOT EXISTS (Vec) THEN
  RETURN(?);

```

```

ELSE
  Result.DirectionRatios[1] := -Vec.DirectionRatios[2];
  Result.DirectionRatios[2] := Vec.DirectionRatios[1];
  RETURN(result);
END_IF;
END_FUNCTION;
FUNCTION IfcScalarTimesVector (
  Scalar : REAL;
  Vec : IfcVectorOrDirection)
  : IfcVector;
LOCAL
  Mag : REAL := 0.0;
  V : IfcDirection
    := IfcGeometricRepresentationItem() || IfcDirection([1.0,0.0,0.0]);
  Result : IfcVector
    := IfcGeometricRepresentationItem() || IfcVector (
      IfcGeometricRepresentationItem() || IfcDirection([1.0,0.0,0.0])
      ,1.0);
END_LOCAL;
IF NOT EXISTS (Scalar) OR NOT EXISTS (Vec) THEN
  Result := ?;
ELSE
  IF 'IFC150FINAL.IFCVECTOR' IN TYPEOF (Vec) THEN
    V := Vec.Orientation;
    Mag := Scalar * Vec.Magnitude;
  ELSE
    V := Vec;
    Mag := Scalar;
  END_IF;
  IF (Mag < 0.0 ) THEN
    REPEAT i := 1 TO SIZEOF(V.DirectionRatios);
      V.DirectionRatios[i] := -V.DirectionRatios[i];
    END_REPEAT;
    Mag := -Mag;
  END_IF;
  Result.Orientation := IfcNormalise(V);
  Result.Magnitude := Mag;
END_IF;
RETURN (Result);
END_FUNCTION;
FUNCTION IfcVectorDifference (
  Arg1, Arg2 : IfcVectorOrDirection)
  : IfcVector;
LOCAL
  Mag, Mag1, Mag2 : REAL := 0.0;
  Ndim : INTEGER := 0;
  Res, Vec1, Vec2 : IfcDirection
    := IfcGeometricRepresentationItem() || IfcDirection([1.0,0.0,0.0]);
  Result : IfcVector
    := IfcGeometricRepresentationItem() || IfcVector (
      IfcGeometricRepresentationItem() || IfcDirection([1.0,0.0,0.0])
      ,1.0);
END_LOCAL;
IF ((NOT EXISTS (Arg1)) OR (NOT EXISTS (Arg2))) OR (Arg1.Dim <> Arg2.Dim) THEN
  Result := ?;
ELSE
  BEGIN
    IF 'IFC150FINAL.IFCVECTOR' IN TYPEOF(Arg1) THEN
      Mag1 := Arg1.Magnitude;
      Vec1 := Arg1.Orientation;
    ELSE
      Mag1 := 1.0;
      Vec1 := Arg1;
    END_IF;
    IF 'IFC150FINAL.IFCVECTOR' IN TYPEOF(Arg2) THEN
      Mag2 := Arg2.Magnitude;
      Vec2 := Arg2.Orientation;
    ELSE

```

```

    Mag2 := 1.0;
    Vec2 := Arg2;
END_IF;
Vec1 := IfcNormalise (Vec1);
Vec2 := IfcNormalise (Vec2);
Ndim := SIZEOF(Vec1.DirectionRatios);
Mag := 0.0;
REPEAT i := 1 TO Ndim;
    Res.DirectionRatios[i] := Mag1 * Vec1.DirectionRatios[i] -
        Mag2 * Vec2.DirectionRatios[i];
    Mag := Mag + (Res.DirectionRatios[i] * Res.DirectionRatios[i]);
END_REPEAT;
IF (Mag > 0.0) THEN
    Result.Magnitude := SQRT(Mag);
    Result.Orientation := Res;
ELSE
    Result.Magnitude := 0.0;
    Result.Orientation := Vec1;
END_IF;
END;
END_IF;
RETURN (Result);
END_FUNCTION;
-- IfcGeometryResource - Function Definition for AttDriven
FUNCTION IfcCircleProfileIntoCurve (
    ProfileDef : IfcCircleProfileDef
    : IfcTrimmedCurve;
LOCAL
    Pos : IfcAxis2Placement2D;
    Circle : IfcCircle;
    ResCurve : IfcTrimmedCurve;
END_LOCAL;
    Pos := ProfileDef.IfAttDrivenProfileDef.Position;
    Circle := IfcGeometricRepresentationItem() || IfcConic(Pos) ||
        IfcCircle(ProfileDef.Radius);
    ResCurve := IfcGeometricRepresentationItem() || IfcBoundedCurve() ||
        IfcTrimmedCurve(
            Circle, [0.0], [2*PI], TRUE, Parameter);
    RETURN (ResCurve);
END_FUNCTION;
FUNCTION IfcRectangleProfileIntoCurve (
    ProfileDef : IfcRectangleProfileDef
    : IfcPolyline;
LOCAL
    Points : LIST [4:4] OF IfcCartesianPoint
        := [IfcGeometricRepresentationItem() ||
            IfcCartesianPoint([0.0,0.0,0.0]);4];
    TempDir : IfcDirection
        := IfcGeometricRepresentationItem() ||
            IfcDirection([1.0,0.0,0.0]);
    ResCurve : IfcPolyline;
END_LOCAL;
    Points[1] := ProfileDef.IfAttDrivenProfileDef.Position.Location;
    TempDir := ProfileDef.IfAttDrivenProfileDef.Position.P[1];
    Points[2] := IfcPointTranslation (Points[1],
        IfcGeometricRepresentationItem() || IfcVector (
            TempDir, ProfileDef.XDim ));
    TempDir := IfcOrthogonalComplement(TempDir);
    Points[3] := IfcPointTranslation (Points[2],
        IfcGeometricRepresentationItem() || IfcVector (
            TempDir, ProfileDef.YDim ));
    TempDir := IfcOrthogonalComplement(TempDir);
    Points[4] := IfcPointTranslation (Points[3],
        IfcGeometricRepresentationItem() || IfcVector (
            TempDir, ProfileDef.XDim ));
    ResCurve := IfcGeometricRepresentationItem() ||
        IfcPolyline([Points[1],Points[2],Points[3],Points[4],Points[1]]);
    RETURN (ResCurve);

```

```

END_FUNCTION;
FUNCTION IfcTrapeziumProfileIntoCurve (
  ProfileDef : IfcTrapeziumProfileDef
  : IfcPolyline;
LOCAL
  Points : LIST [4:4] OF IfcCartesianPoint
  := [IfcGeometricRepresentationItem() ||
    IfcCartesianPoint([0.0,0.0]);4];
  TempDir : IfcDirection
  := IfcGeometricRepresentationItem() ||
    IfcDirection([1.0,0.0]);
  TempPoint : IfcCartesianPoint
  := IfcGeometricRepresentationItem() ||
    IfcCartesianPoint([0.0,0.0]);
  ResCurve : IfcPolyline;
END_LOCAL;
Points[1] := ProfileDef.IfctrapeziumProfileDef.Position.Location;
TempDir := ProfileDef.IfctrapeziumProfileDef.Position.P[1];
Points[2] := IfcPointTranslation (Points[1],
  IfcGeometricRepresentationItem() || IfcVector (
    TempDir, ProfileDef.BottomXDim ));
TempDir := IfcOrthogonalComplement(TempDir);
TempPoint := IfcPointTranslation (Points[2],
  IfcGeometricRepresentationItem() || IfcVector (
    TempDir, ProfileDef.YDim ));
TempDir := IfcOrthogonalComplement(TempDir);
Points[3] := IfcPointTranslation (TempPoint,
  IfcGeometricRepresentationItem() || IfcVector (
    TempDir,
    (ProfileDef.BottomXDim - ProfileDef.TopXDim - ProfileDef.TopXOffset)));
Points[4] := IfcPointTranslation (Points[3],
  IfcGeometricRepresentationItem() || IfcVector (
    TempDir, ProfileDef.TopXDim ));
ResCurve := IfcGeometricRepresentationItem() ||
  IfcPolyline([Points[1],Points[2],Points[3],Points[4],Points[1]]);
RETURN (ResCurve);
END_FUNCTION;
FUNCTION IfcProfileIntoArea (
  ProfileDef : IfcAttDrivenProfileDef
  : IfcCurveBoundedPlane;
LOCAL
  Curve2D : Ifc2DCompositeCurve;
  ResSurface : IfcCurveBoundedPlane;
END_LOCAL;
Curve2D := IfcGeometricRepresentationItem() ||
  IfcCurve() ||
  IfcBoundedCurve() ||
  IfcCompositeCurve(
    [IfcCompositeCurveSegment (Continuous,
      TRUE, ProfileDef.CurveForSurface)], FALSE) ||
  Ifc2DCompositeCurve();
ResSurface := IfcGeometricRepresentationItem() ||
  IfcSurface() ||
  IfcCurveBoundedPlane(
    IfcGeometricRepresentationItem() || IfcSurface() || IfcElementarySurface(
      IfcGeometricRepresentationItem() || IfcPlacement(
        IfcGeometricRepresentationItem() ||
        IfcCartesianPoint([0.0, 0.0, 0.0])) ||
        IfcAxis2Placement3D(
          IfcGeometricRepresentationItem() || IfcDirection([0.0,0.0,1.0]),
          IfcGeometricRepresentationItem() || IfcDirection([1.0,0.0,0.0])) ||
        IfcPlane(),
        Curve2D, []);
RETURN (ResSurface);
END_FUNCTION;
FUNCTION IfcComputeAngleFromAxis (
  Dir : IfcDirection
  : IfcPlaneAngleMeasure;

```



```

LOCAL
  Angle := IfcPlaneAngleMeasure;
END_LOCAL;
IF (Dir.Dim <> 2) OR (NOT EXISTS (Dir)) THEN
  RETURN (?);
END_IF;
Angle := ATAN(Dir.DirectionRatios[2], Dir.DirectionRatios[1]);
IF (Dir.DirectionRatios[1] < 0.0) THEN
  Angle := Angle + PI;
ELSE
  IF (Dir.DirectionRatios[2] < 0.0) THEN
    Angle := Angle + 2 * PI;
  END_IF;
END_IF;
RETURN (Angle);
END_FUNCTION;
FUNCTION IfcPointTranslation (
  Origin : IfcCartesianPoint;
  Vec    : IfcVector)
  : IfcCartesianPoint;
LOCAL
  NDim : INTEGER := HIINDEX(Origin.Coordinates);
  Point : IfcCartesianPoint
    := IfcGeometricRepresentationItem() ||
      IfcCartesianPoint([0.0,0.0]);
END_LOCAL;
IF (Origin.Dim <> Vec.Dim) OR (NOT EXISTS (Vec)) OR (NOT EXISTS (Origin)) THEN
  RETURN (?);
END_IF;
REPEAT i := 1 TO NDim;
  Point.Coordinates[i] := Origin.Coordinates[i] +
    Vec.Magnitude * Vec.Orientation.DirectionRatios[i];
END_REPEAT;
RETURN (Point);
END_FUNCTION;
FUNCTION IfcExtrusionPath (
  Solid : IfcAttDrivenExtrudedSolid)
  : IfcPolyline;
LOCAL
  Path : IfcPolyline
    := IfcGeometricRepresentationItem() || IfcCurve() || IfcBoundedCurve() ||
      IfcPolyline([IfcGeometricRepresentationItem() || IfcPoint() || IfcCartesianPoint([0.0,0.0,0.0]),
        IfcGeometricRepresentationItem() || IfcPoint() || IfcCartesianPoint([0.0,0.0,1.0])]);
  Depth : IfcPositiveLengthMeasure := 0;
  NDim : INTEGER := HIINDEX(Solid.Segments);
END_LOCAL;
REPEAT i := 1 TO NDim;
  Depth := Depth + Solid.Segments[i].Depth;
END_REPEAT;
Path := IfcGeometricRepresentationItem() || IfcCurve() || IfcBoundedCurve() ||
  IfcPolyline([Solid.Segments[1].Position.Location,
    IfcPointTranslation (Solid.Segments[1].Position.Location,
      IfcGeometricRepresentationItem() || IfcVector (
        Solid.Segments[1].Position.Axis, Depth)]));
RETURN (Path);
END_FUNCTION;
FUNCTION IfcRevolutionPath (
  Solid : IfcAttDrivenRevolvedSolid)
  : IfcTrimmedCurve;
LOCAL
  Path : IfcTrimmedCurve;
  Pos : IfcAxis2Placement3D;
  Circle : IfcCircle;
  Angle : IfcPlaneAngleMeasure := 0;
  NDim : INTEGER := HIINDEX(Solid.Segments);
END_LOCAL;
REPEAT i := 1 TO NDim;
  Angle := Angle + Solid.Segments[i].Angle;

```

```

END_REPEAT;
Pos := IfcGeometricRepresentationItem() || IfcPlacement (
    IfcGeometricRepresentationItem() || IfcPoint() || IfcCartesianPoint([
        (Solid.Segments[1].Position.Location.Coordinates[1] + Solid.Segments[1].Axis.Location.Coordinates[1])
        ,(Solid.Segments[1].Position.Location.Coordinates[2] + Solid.Segments[1].Axis.Location.Coordinates[1])
        ,(Solid.Segments[1].Position.Location.Coordinates[3] + Solid.Segments[1].Axis.Location.Coordinates[1]))] ||
    IfcAxis2Placement3D (
        IfcGeometricRepresentationItem() || IfcDirection([0.0, 1.0, 0.0]),
        IfcGeometricRepresentationItem() || IfcDirection([-1.0, 0.0, 0.0]));
Circle := IfcGeometricRepresentationItem() || IfcConic(Pos) ||
    IfcCircle(Solid.Segments[1].Axis.Location.Coordinates[1]);
Path := IfcGeometricRepresentationItem() || IfcBoundedCurve() ||
    IfcTrimmedCurve(
        Circle, [0.0], [Angle], TRUE, Parameter);
RETURN (Path);
END_FUNCTION;
-- IfcKernel - Type Definitions
TYPE IfcProxyTypeEnum = ENUMERATION OF (
    Product
    ,Process
    ,Control
    ,Document
    ,Resource );
END_TYPE;
TYPE IfcSequenceTypeEnum = ENUMERATION OF (
    Start_Start
    ,Start_Finish
    ,Finish_Start
    ,Finish_Finish );
END_TYPE;
TYPE IfcObjectWithPlacementSelect = SELECT (
    IfcProduct
    ,IfcModelingAid
    ,IfcProject);
END_TYPE;
-- IfcKernel - Entity Definitions
ENTITY IfcControl
ABSTRACT SUPERTYPE OF (ONEOF (IfcConnectionGeometry,IfcCostElement,IfcSpaceProgram,IfcWorkSchedule))
SUBTYPE OF (IfcObject);
END_ENTITY;
ENTITY IfcDocument
ABSTRACT SUPERTYPE
SUBTYPE OF (IfcObject);
END_ENTITY;
ENTITY IfcGroup
SUPERTYPE OF (ONEOF (IfcCostElementGroup,IfcSpaceProgramGroup,IfcSystem,IfcWorkGroup,IfcZone))
SUBTYPE OF (IfcObject);
    GroupPurpose : OPTIONAL STRING;
INVERSE
    GroupedBy : IfcRelGroups FOR RelatingObject;
END_ENTITY;
ENTITY IfcLocalPlacement
SUBTYPE OF (IfcModelingAid);
    PlacementRelTo : OPTIONAL IfcObjectWithPlacementSelect;
    RelativePlacement : IfcAxis2Placement;
END_ENTITY;
ENTITY IfcModelingAid
ABSTRACT SUPERTYPE OF (ONEOF
(IfcLocalPlacement,IfcDesignGrid,IfcGridAxis,IfcGridIntersection,IfcGridLevel,IfcPlacementConstraint,IfcReferenceGeometryAid))
SUBTYPE OF (IfcRoot);
END_ENTITY;
ENTITY IfcObject
ABSTRACT SUPERTYPE OF (ONEOF(
    IfcProduct
    ,IfcProcess
    ,IfcControl
    ,IfcDocument
    ,IfcGroup

```

```

,IfcProject
,IfcProxy
,IfcResource))
SUBTYPE OF (IfcRoot);
  OwnerHistory      : IfcOwnerHistory;
  TypeDefinitions   : LIST [0:?] OF IfcPropertyTypeDef;
  OccurrenceProperties : LIST [0:?] OF IfcOccurrencePropertySet;
  ExtendedProperties : LIST [0:?] OF IfcPropertySet;
INVERSE
  PartOfGroups      : SET [0:?] OF IfcRelGroups
                    FOR RelatedObjects;
WHERE
  WR1: SIZEOF(QUERY(temp <= ExtendedProperties |
    ('Ifc150FINAL.IFCOCCURRENCEPROPERTYSET' IN TYPEOF(temp))
    OR
    ('Ifc150FINAL.IFCSHAREDPROPERTYSET' IN TYPEOF(temp))))
    = 0;
END_ENTITY;
ENTITY IfcProcess
ABSTRACT SUPERTYPE
SUBTYPE OF (IfcObject);
  PerformedBy      : SET [0:?] OF IfcActorSelect;
  Classification    : OPTIONAL IfcClassificationList;
INVERSE
  IsSuccessorFrom   : SET [0:?] OF IfcRelSequence
                    FOR RelatedObject;
  IsPredecessorTo   : SET [0:?] OF IfcRelSequence
                    FOR RelatingObject;
  UsesProducts      : SET [0:2] OF IfcRelUsesProducts
                    FOR RelatingObject;
END_ENTITY;
ENTITY IfcProduct
ABSTRACT SUPERTYPE OF (ONEOF (IfcBuilding,IfcBuildingStorey,IfcElement,IfcSite,IfcSpatialElement))
SUBTYPE OF (IfcObject);
  LocalPlacement    : IfcLocalPlacement;
  ProductShape      : OPTIONAL IfcProductShape;
  ProductCost       : OPTIONAL IfcCost;
  Classification     : OPTIONAL IfcClassificationList;
INVERSE
  UsedInProcesses   : SET [0:?] OF IfcRelUsesProducts
                    FOR RelatedObjects;
END_ENTITY;
ENTITY IfcProject
SUBTYPE OF (IfcObject);
  GlobalId          : OPTIONAL IfcGloballyUniqueId;
  UnitsInContext    : IfcUnitAssignment;
  ProjectTeam       : IfcProjectTeamRegistry;
  ProjectApps       : IfcProjectAppRegistry;
  Classification     : OPTIONAL IfcClassificationList;
  AbsolutePlacement : IfcAxis2Placement;
END_ENTITY;
ENTITY IfcProxy
SUBTYPE OF (IfcObject);
  ProxyType         : IfcProxyTypeEnum;
  LocalPlacement    : OPTIONAL IfcLocalPlacement;
  ProductShape      : OPTIONAL IfcProductShape;
WHERE
  WR1: NOT(EXISTS(SELF\IfcObject.TypeDefinitions));
  WR2: HINDEX(SELF\IfcObject.OccurrenceProperties) = 0;
  WR3: ((ProxyType = IfcProxyTypeEnum.Product) AND (EXISTS(LocalPlacement)))
    OR
    ((ProxyType <> IfcProxyTypeEnum.Product) AND NOT(EXISTS(LocalPlacement)));
END_ENTITY;
ENTITY IfcRelationship
ABSTRACT SUPERTYPE OF (ONEOF(
  IfcRelationship1toN
  ,IfcRelationship1to1))
SUBTYPE OF (IfcRoot);

```

```

    OwnerHistory      : IfcOwnerHistory;
    ExtendedProperties : LIST [0:?] OF IfcPropertySet;
    RelatedIsDependent : BOOLEAN;
    RelatingIsDependent : BOOLEAN;
END_ENTITY;
ENTITY IfcRelationship1to1
ABSTRACT SUPERTYPE OF (ONEOF (IfcRelSequence,IfcRelAdjacencyReq,IfcRelConnectsElements,IfcRelResourceUse))
SUBTYPE OF (IfcRelationship);
    RelatingObject      : IfcObject;
    RelatedObject        : IfcObject;
WHERE
    WR1: RelatedObject <>: RelatingObject;
END_ENTITY;
ENTITY IfcRelationship1toN
ABSTRACT SUPERTYPE OF (ONEOF
(IfcRelGroups,IfcRelUsesProducts,IfcRelAssemblesElements,IfcRelAssemblesSpaces,IfcRelContains,IfcRelCostScheduleElement,Ifc
RelCoversBldgElements,IfcRelFillsElements,IfcRelSeparatesSpaces,IfcRelServicesBuildings,IfcRelVoidsElements))
SUBTYPE OF (IfcRelationship);
    RelatingObject      : IfcObject;
    RelatedObjects       : LIST [1:?] OF IfcObject;
WHERE
    WR1: SIZEOF(QUERY(Temp < * RelatedObjects |
        RelatingObject := Temp)) = 0;
END_ENTITY;
ENTITY IfcRelGroups
SUBTYPE OF (IfcRelationship1toN);
-- SELF\IfcRelationship1toN.RelatingObject : IfcGroup;
-- SELF\IfcRelationship1toN.RelatedObjects : LIST [1:?] OF IfcObject;
END_ENTITY;
ENTITY IfcRelSequence
SUBTYPE OF (IfcRelationship1to1);
-- SELF\IfcRelationship1to1.RelatingObject : IfcProcess;
-- SELF\IfcRelationship1to1.RelatedObject : IfcProcess;
    TimeLag              : IfcTimeDurationMeasure;
    SequenceType          : IfcSequenceTypeEnum;
END_ENTITY;
ENTITY IfcRelUsesProducts
SUBTYPE OF (IfcRelationship1toN);
-- SELF\IfcRelationship1toN.RelatingObject : IfcProcess;
-- SELF\IfcRelationship1toN.RelatedObjects : LIST [1:?] OF IfcProduct;
    InOrOut : LOGICAL;
END_ENTITY;
ENTITY IfcResource
SUBTYPE OF (IfcObject);
END_ENTITY;
ENTITY IfcRoot
ABSTRACT SUPERTYPE OF (ONEOF(
    IfcObject
    ,IfcRelationship
    ,IfcModelingAid));
    ProjectId      : IfcProjectUniqueId;
UNIQUE
    UR1: ProjectId;
END_ENTITY;
-- IfcMeasureResource - Type Definition for Unit and Measure
TYPE IfcAmountOfSubstanceMeasure = REAL;
END_TYPE;
TYPE IfcAreaMeasure = REAL;
END_TYPE;
TYPE IfcContextDependentMeasure = REAL;
END_TYPE;
TYPE IfcCountMeasure = NUMBER;
END_TYPE;
TYPE IfcDescriptiveMeasure = STRING;
END_TYPE;
TYPE IfcElectricCurrentMeasure = REAL;
END_TYPE;
TYPE IfcLengthMeasure = REAL;

```

```

END_TYPE;
TYPE IfcLuminousIntensityMeasure = REAL;
END_TYPE;
TYPE IfcMassMeasure = REAL;
END_TYPE;
TYPE IfcNumericMeasure = NUMBER;
END_TYPE;
TYPE IfcParameterValue = REAL;
END_TYPE;
TYPE IfcPositiveLengthMeasure = IfcLengthMeasure;
WHERE
  WR1: SELF > 0;
END_TYPE;
TYPE IfcPositivePlaneAngleMeasure = IfcPlaneAngleMeasure;
WHERE
  WR1: SELF > 0;
END_TYPE;
TYPE IfcPositiveRatioMeasure = IfcRatioMeasure;
WHERE
  WR1: SELF > 0;
END_TYPE;
TYPE IfcPlaneAngleMeasure = REAL;
END_TYPE;
TYPE IfcRatioMeasure = REAL;
END_TYPE;
TYPE IfcSolidAngleMeasure = REAL;
END_TYPE;
TYPE IfcThermodynamicTemperatureMeasure = REAL;
END_TYPE;
TYPE IfcTimeMeasure = REAL;
END_TYPE;
TYPE IfcDurationMeasure = REAL;
END_TYPE;
TYPE IfcTimeStamp = INTEGER;
END_TYPE;
TYPE IfcVolumeMeasure = REAL;
END_TYPE;
TYPE IfcCompoundPlaneAngleMeasure = LIST [3:3] OF INTEGER;
WHERE
  WR1: { 0 <= SELF[1] < 360 };
  WR2: { 0 <= SELF[2] < 60 };
  WR3: { 0 <= SELF[3] < 60 };
END_TYPE;
TYPE IfcBoolean = BOOLEAN;
END_TYPE;
TYPE IfcInteger = INTEGER;
END_TYPE;
TYPE IfcReal = REAL;
END_TYPE;
TYPE IfcString = STRING;
END_TYPE;
TYPE IfcSiPrefix = ENUMERATION OF (
  EXA
  ,PETA
  ,TERA
  ,GIGA
  ,MEGA
  ,KILO
  ,HECTO
  ,DECA
  ,DECI
  ,CENTI
  ,MILLI
  ,MICRO
  ,NANO
  ,PICO
  ,FEMTO
  ,ATTO );

```

```

END_TYPE;
TYPE IfcSiUnitName = ENUMERATION OF (
    METRE
    ,SQUARE_METRE
    ,CUBIC_METRE
    ,GRAM
    ,SECOND
    ,AMPERE
    ,KELVIN
    ,MOLE
    ,CANDELA
    ,RADIAN
    ,STERADIAN
    ,HERTZ
    ,NEWTON
    ,PASCAL
    ,JOULE
    ,WATT
    ,COULOMB
    ,VOLT
    ,FARAD
    ,OHM
    ,SIEMENS
    ,WEBER
    ,TESLA
    ,HENRY
    ,DEGREE_CELSIUS
    ,LUMEN
    ,LUX
    ,BECQUEREL
    ,GRAY
    ,SIEVERT);
END_TYPE;
TYPE IfcDerivedUnitTypeEnum = ENUMERATION OF (
    VolumetricFlowrateUnit
    ,MassFlowrateUnit
    ,PressureUnit
    ,EnergyUnit
    ,PowerUnit
    ,AngularVelocityUnit
    ,LinearVelocityUnit
    ,RotationalFrequencyUnit
    ,HeatfluxDensityUnit
    ,MassDensityUnit
    ,ThermalResistanceUnit
    ,ThermalTransmittanceUnit
    ,VoltageUnit
    ,DynamicViscosityUnit
    ,KinematicViscosityUnit
    ,Unspecified);
END_TYPE;
TYPE IfcUnitTypeEnum = ENUMERATION OF (
    LengthUnit
    ,MassUnit
    ,TimeUnit
    ,DurationUnit
    ,ElectricCurrentUnit
    ,ThermodynamicTemperatureUnit
    ,AmountOfSubstanceUnit
    ,LuminousIntensityUnit
    ,PlaneAngleUnit
    ,SolidAngleUnit
    ,AreaUnit
    ,VolumeUnit
    ,RatioUnit
    ,Unspecified );
END_TYPE;
TYPE IfcMeasureValue = SELECT (

```

```

    ,IfcLengthMeasure
    ,IfcMassMeasure
    ,IfcTimeMeasure
    ,IfcElectricCurrentMeasure
    ,IfcThermodynamicTemperatureMeasure
    ,IfcAmountOfSubstanceMeasure
    ,IfcLuminousIntensityMeasure
    ,IfcPlaneAngleMeasure
    ,IfcSolidAngleMeasure
    ,IfcAreaMeasure
    ,IfcVolumeMeasure
    ,IfcRatioMeasure
    ,IfcParameterValue
    ,IfcNumericMeasure
    ,IfcContextDependentMeasure
    ,IfcDescriptiveMeasure
    ,IfcPositiveLengthMeasure
    ,IfcPositivePlaneAngleMeasure
    ,IfcPositiveRatioMeasure
    ,IfcCountMeasure
    ,IfcCompoundPlaneAngleMeasure
    ,IfcTimeDurationMeasure
    ,IfcTimeStamp);
END_TYPE;
TYPE IfcUnit = SELECT (
    ,IfcDerivedUnit
    ,IfcNamedUnit );
END_TYPE;
-- IfcMeasureResource - Entity Definition for Unit and Measure
ENTITY IfcConversionBasedUnit
    SUBTYPE OF (IfcNamedUnit);
    Name : STRING;
    ConversionFactor : IfcMeasureWithUnit;
END_ENTITY;
ENTITY IfcContextDependentUnit
    SUBTYPE OF (IfcNamedUnit);
    Name : STRING;
END_ENTITY;
ENTITY IfcDerivedUnit;
    Elements : SET [1:?] OF IfcDerivedUnitElement;
    UnitType : IfcDerivedUnitTypeEnum;
DERIVE
    Dimensions : IfcDimensionalExponents
        := IfcDeriveDimensionalExponents(SELF);
WHERE
    WR1: (SIZEOF (Elements) > 1)
        OR ((SIZEOF (Elements) = 1)
            AND (Elements[1].Exponent <> 1 ));
END_ENTITY;
ENTITY IfcDerivedUnitElement;
    Unit : IfcNamedUnit;
    Exponent : INTEGER;
END_ENTITY;
ENTITY IfcDimensionalExponents;
    LengthExponent : INTEGER;
    MassExponent : INTEGER;
    TimeExponent : INTEGER;
    ElectricCurrentExponent : INTEGER;
    ThermodynamicTemperatureExponent : INTEGER;
    AmountOfSubstanceExponent : INTEGER;
    LuminousIntensityExponent : INTEGER;
END_ENTITY;
ENTITY IfcMeasureWithUnit;
    ValueComponent : IfcMeasureValue;
    UnitComponent : IfcUnit;
END_ENTITY;
ENTITY IfcNamedUnit
    SUPERTYPE OF (ONEOF(

```

```

IfcSiUnit
,IfcConversionBasedUnit
,IfcContextDependentUnit ));
Dimensions : IfcDimensionalExponents;
UnitType : IfcUnitTypeEnum;
WHERE
WR1: IfcCorrectDimensions
( SELF.UnitType, SELF.Dimensions);
END_ENTITY;
ENTITY IfcSiUnit
SUBTYPE OF (IfcNamedUnit);
Prefix : OPTIONAL IfcSiPrefix;
Name : IfcSiUnitName;
DERIVE
SELF\IfcNamedUnit.Dimensions
: IfcDimensionalExponents
:= IfcDimensionsForSiUnit (SELF.Name);
END_ENTITY;
ENTITY IfcUnitAssignment;
Units : SET [1:?] OF IfcUnit;
END_ENTITY;
-- IfcMeasureResource - Function Definition for Unit and Measure
FUNCTION IfcDeriveDimensionalExponents (
x : IfcUnit)
: IfcDimensionalExponents;
LOCAL
Result : IfcDimensionalExponents :=
IfcDimensionalExponents(0, 0, 0, 0, 0, 0, 0);
END_LOCAL;
IF 'IFC150FINAL.IFCDERIVEDUNIT' IN TYPEOF(x) THEN
REPEAT i := LOINDEX(x.Elements) TO HIINDEX(x.Elements);
Result.LengthExponent :=
Result.LengthExponent +
(x.Elements[i].Exponent *
x.Elements[i].Unit.Dimensions.LengthExponent);
Result.MassExponent :=
Result.MassExponent +
(x.Elements[i].Exponent *
x.Elements[i].Unit.Dimensions.MassExponent);
Result.TimeExponent :=
Result.TimeExponent +
(x.Elements[i].Exponent *
x.Elements[i].Unit.Dimensions.TimeExponent);
Result.ElectricCurrentExponent :=
Result.ElectricCurrentExponent +
(x.Elements[i].Exponent *
x.Elements[i].Unit.Dimensions.ElectricCurrentExponent);
Result.ThermodynamicTemperatureExponent :=
Result.ThermodynamicTemperatureExponent +
(x.Elements[i].Exponent *
x.Elements[i].Unit.Dimensions.ThermodynamicTemperatureExponent);
Result.AmountOfSubstanceExponent :=
Result.AmountOfSubstanceExponent +
(x.Elements[i].Exponent *
x.Elements[i].Unit.Dimensions.AmountOfSubstanceExponent);
Result.LuminousIntensityExponent :=
Result.LuminousIntensityExponent +
(x.Elements[i].Exponent *
x.Elements[i].Unit.Dimensions.LuminousIntensityExponent);
END_REPEAT;
ELSE -- x is a unitless or a named unit
Result := x.Dimensions;
END_IF;
RETURN (Result);
END_FUNCTION;
FUNCTION IfcDimensionsForSiUnit
(n : IfcSiUnitName )
: IfcDimensionalExponents;

```



```

CASE n OF
  METRE      : RETURN (IfcDimensionalExponents
    (1, 0, 0, 0, 0, 0, 0));
  SQUARE_METRE : RETURN (IfcDimensionalExponents
    (2, 0, 0, 0, 0, 0, 0));
  CUBIC_METRE  : RETURN (IfcDimensionalExponents
    (3, 0, 0, 0, 0, 0, 0));
  GRAM         : RETURN (IfcDimensionalExponents
    (0, 1, 0, 0, 0, 0, 0));
  SECOND       : RETURN (IfcDimensionalExponents
    (0, 0, 1, 0, 0, 0, 0));
  AMPERE       : RETURN (IfcDimensionalExponents
    (0, 0, 0, 1, 0, 0, 0));
  KELVIN       : RETURN (IfcDimensionalExponents
    (0, 0, 0, 0, 1, 0, 0));
  MOLE         : RETURN (IfcDimensionalExponents
    (0, 0, 0, 0, 0, 1, 0));
  CANDELA     : RETURN (IfcDimensionalExponents
    (0, 0, 0, 0, 0, 0, 1));
  RADIAN       : RETURN (IfcDimensionalExponents
    (0, 0, 0, 0, 0, 0, 0));
  STERADIAN    : RETURN (IfcDimensionalExponents
    (0, 0, 0, 0, 0, 0, 0));
  HERTZ        : RETURN (IfcDimensionalExponents
    (0, 0, -1, 0, 0, 0, 0));
  NEWTON       : RETURN (IfcDimensionalExponents
    (1, 1, -2, 0, 0, 0, 0));
  PASCAL       : RETURN (IfcDimensionalExponents
    (-1, 1, -2, 0, 0, 0, 0));
  JOULE        : RETURN (IfcDimensionalExponents
    (2, 1, -2, 0, 0, 0, 0));
  WATT         : RETURN (IfcDimensionalExponents
    (2, 1, -3, 0, 0, 0, 0));
  COULOMB      : RETURN (IfcDimensionalExponents
    (0, 0, 1, 1, 0, 0, 0));
  VOLT         : RETURN (IfcDimensionalExponents
    (2, 1, -3, -1, 0, 0, 0));
  FARAD        : RETURN (IfcDimensionalExponents
    (-2, -1, 4, 1, 0, 0, 0));
  OHM          : RETURN (IfcDimensionalExponents
    (2, 1, -3, -2, 0, 0, 0));
  SIEMENS      : RETURN (IfcDimensionalExponents
    (-2, -1, 3, 2, 0, 0, 0));
  WEBER        : RETURN (IfcDimensionalExponents
    (2, 1, -2, -1, 0, 0, 0));
  TESLA        : RETURN (IfcDimensionalExponents
    (0, 1, -2, -1, 0, 0, 0));
  HENRY        : RETURN (IfcDimensionalExponents
    (2, 1, -2, -2, 0, 0, 0));
  DEGREE_CELSIUS : RETURN (IfcDimensionalExponents
    (0, 0, 0, 0, 1, 0, 0));
  LUMEN        : RETURN (IfcDimensionalExponents
    (0, 0, 0, 0, 0, 0, 1));
  LUX          : RETURN (IfcDimensionalExponents
    (-2, 0, 0, 0, 0, 0, 1));
  BECQUEREL    : RETURN (IfcDimensionalExponents
    (0, 0, -1, 0, 0, 0, 0));
  GRAY         : RETURN (IfcDimensionalExponents
    (2, 0, -2, 0, 0, 0, 0));
  SIEVERT      : RETURN (IfcDimensionalExponents
    (2, 0, -2, 0, 0, 0, 0));
  OTHERWISE    : RETURN (IfcDimensionalExponents
    (0, 0, 0, 0, 0, 0, 0));
END_CASE;
END_FUNCTION;
FUNCTION IfcCorrectDimensions
(m : IfcUnitTypeEnum;
 Dim : IfcDimensionalExponents)

```

```

: LOGICAL;
CASE m OF
LengthUnit : IF
  Dim = (IfcDimensionalExponents
    (1, 0, 0, 0, 0, 0, 0))
  THEN RETURN(TRUE);
  ELSE RETURN(FALSE);
END_IF;
MassUnit : IF
  Dim = (IfcDimensionalExponents
    (0, 1, 0, 0, 0, 0, 0))
  THEN RETURN(TRUE);
  ELSE RETURN(FALSE);
END_IF;
TimeUnit : IF
  Dim = (IfcDimensionalExponents
    (0, 0, 1, 0, 0, 0, 0))
  THEN RETURN(TRUE);
  ELSE RETURN(FALSE);
END_IF;
ElectricCurrentUnit : IF
  Dim = (IfcDimensionalExponents
    (0, 0, 0, 1, 0, 0, 0))
  THEN RETURN(TRUE);
  ELSE RETURN(FALSE);
END_IF;
ThermodynamicTemperatureUnit : IF
  Dim = (IfcDimensionalExponents
    (0, 0, 0, 0, 1, 0, 0))
  THEN RETURN(TRUE);
  ELSE RETURN(FALSE);
END_IF;
AmountOfSubstanceUnit : IF
  Dim = (IfcDimensionalExponents
    (0, 0, 0, 0, 0, 1, 0))
  THEN RETURN(TRUE);
  ELSE RETURN(FALSE);
END_IF;
LuminousIntensityUnit : IF
  Dim = (IfcDimensionalExponents
    (0, 0, 0, 0, 0, 0, 1))
  THEN RETURN(TRUE);
  ELSE RETURN(FALSE);
END_IF;
PlaneAngleUnit : IF
  Dim = (IfcDimensionalExponents
    (0, 0, 0, 0, 0, 0, 0))
  THEN RETURN(TRUE);
  ELSE RETURN(FALSE);
END_IF;
SolidAngleUnit : IF
  Dim = (IfcDimensionalExponents
    (0, 0, 0, 0, 0, 0, 0))
  THEN RETURN(TRUE);
  ELSE RETURN(FALSE);
END_IF;
AreaUnit : IF
  Dim = (IfcDimensionalExponents
    (2, 0, 0, 0, 0, 0, 0))
  THEN RETURN(TRUE);
  ELSE RETURN(FALSE);
END_IF;
VolumeUnit : IF
  Dim = (IfcDimensionalExponents
    (3, 0, 0, 0, 0, 0, 0))
  THEN RETURN(TRUE);
  ELSE RETURN(FALSE);
END_IF;

```

```

RatioUnit : IF
  Dim = (IfcDimensionalExponents
    (0, 0, 0, 0, 0, 0, 0))
  THEN RETURN(TRUE);
  ELSE RETURN(FALSE);
END_IF;
OTHERWISE :
  RETURN (UNKNOWN);
END_CASE;
END_FUNCTION;
-- IfcModelingAidExtension - Cross schema references --
-- IfcModelingAidExtension - Type definitions --
TYPE IfcReferenceCurveSelect = SELECT
  (IfcReferenceCurve,
   IfcGridAxis);
END_TYPE;
TYPE IfcReferencePointSelect = SELECT
  (IfcGridIntersection,
   IfcReferencePoint);
END_TYPE;
-- IfcModelingAidExtension - Entity definitions --
ENTITY IfcConstrainedPlacement
  SUBTYPE OF (IfcLocalPlacement);
  PathEndPointConstraint : LIST [1:2] OF IfcPlacementConstraint;
END_ENTITY;
ENTITY IfcConstraintRelIntersection
  SUBTYPE OF (IfcPlacementConstraint);
  RefPointAt : IfcReferencePointSelect;
  OffsetFromCurves : LIST [0:3] OF IfcReferenceCurveSelect;
  OffsetDistances : LIST [0:3] OF IfcLengthMeasure;
END_ENTITY;
ENTITY IfcDesignGrid
  SUBTYPE OF (IfcModelingAid);
  GridPurpose : STRING;
  LocalPlacement : IfcLocalPlacement;
  INVERSE
    HasGridLevels : SET[1:?] OF IfcGridLevel FOR PartOfDesignGrid;
END_ENTITY;
ENTITY IfcGridAxis
  SUBTYPE OF (IfcModelingAid);
  PartOfGridLevel : IfcGridLevel;
  AxisTag : STRING;
  AxisCurve : IfcBoundedCurve;
  SameSense : BOOLEAN;
  INVERSE
    AlignedGridIntersections : SET[0:?] OF IfcGridIntersection FOR AlignedWithAxes;
END_ENTITY;
ENTITY IfcGridIntersection
  SUBTYPE OF (IfcModelingAid);
  AlignedWithAxes : SET [2:?] OF IfcGridAxis;
  IntersectionPoint : IfcCartesianPoint;
END_ENTITY;
ENTITY IfcGridLevel
  SUBTYPE OF (IfcModelingAid);
  PartOfDesignGrid : IfcDesignGrid;
  GridLevelHeight : IfcLengthMeasure;
  GridLevelName : STRING;
  INVERSE
    HasGridAxes : SET[1:?] OF IfcGridAxis FOR PartOfGridLevel;
END_ENTITY;
ENTITY IfcPlacementConstraint
  ABSTRACT SUPERTYPE OF (IfcConstraintRelIntersection)
  SUBTYPE OF (IfcModelingAid);
END_ENTITY;
ENTITY IfcReferenceCurve
  SUBTYPE OF (IfcReferenceGeometryAid);
  ReferenceCurve : IfcBoundedCurve;
END_ENTITY;

```

```

ENTITY IfcReferencePoint
    SUBTYPE OF (IfcReferenceGeometryAid);
    ReferencePoint : IfcCartesianPoint;
END_ENTITY;
ENTITY IfcReferenceSurface
    SUBTYPE OF (IfcReferenceGeometryAid);
    ReferenceSurface : IfcSurface;
END_ENTITY;
ENTITY IfcReferenceGeometryAid
    SUPERTYPE OF (ONEOF(IfcReferenceSurface,IfcReferenceCurve,IfcReferencePoint))
    SUBTYPE OF (IfcModelingAid);
    LocalPlacement : IfcLocalPlacement;
END_ENTITY;
-- IfcProcessExtension - Type Definitions
TYPE IfcWorkTaskOrGroupSelect = SELECT (
    IfcWorkTask
    ,IfcWorkGroup);
END_TYPE;
TYPE IfcTaskStatusEnum = ENUMERATION OF (
    Completed
    ,Started
    ,NotYetStarted);
END_TYPE;
-- IfcProcessExtension - Entity Definitions
ENTITY IfcRelResourceUse
    SUBTYPE OF (IfcRelationship1To1);
    ResourceQuantity : LIST [1:?] OF REAL;
    ResourceDuration : IfcTimeDurationMeasure;
-- SELF\IfcRelationship1to1.RelatingObject : IfcProcess;
-- SELF\IfcRelationship1to1.RelatedObject : IfcResource;
END_ENTITY;
ENTITY IfcWorkGroup
    SUBTYPE OF (IfcGroup);
    WorkGroupName : OPTIONAL STRING;
    WorkGroupId : OPTIONAL STRING;
WHERE
    WR1: SIZEOF(QUERY(TEMP <* SELF\IfcGroup.GroupedBy.RelatedObjects |
        'IFC150FINAL.IFCWORKTASK' IN TYPEOF(Temp))) >= 1;
END_ENTITY;
ENTITY IfcWorkSchedule
    SUBTYPE OF (IfcControl);
    WorkSchedule : IfcWorkTaskOrGroupSelect;
    ActualStart : OPTIONAL IfcDateTimeSelect;
    EarliestStart : OPTIONAL IfcDateTimeSelect;
    LatestStart : OPTIONAL IfcDateTimeSelect;
    ActualFinish : OPTIONAL IfcDateTimeSelect;
    EarliestFinish : OPTIONAL IfcDateTimeSelect;
    LatestFinish : OPTIONAL IfcDateTimeSelect;
    StatusTime : OPTIONAL IfcDateTimeSelect;
    ScheduledStart : OPTIONAL IfcDateTimeSelect;
    ScheduledFinish : OPTIONAL IfcDateTimeSelect;
    ScheduledDuration : OPTIONAL IfcTimeDurationMeasure;
    RemainingTime : OPTIONAL IfcTimeDurationMeasure;
    FreeFloat : OPTIONAL IfcTimeDurationMeasure;
    TotalFloat : OPTIONAL IfcTimeDurationMeasure;
    TaskStatus : OPTIONAL IfcTaskStatusEnum;
    IsCritical : OPTIONAL BOOLEAN;
END_ENTITY;
ENTITY IfcWorkTask
    SUBTYPE OF (IfcProcess);
    WorkTaskId : OPTIONAL STRING;
    TaskDescription : OPTIONAL STRING;
    TaskCost : OPTIONAL IfcCost;
    WorkMethod : OPTIONAL STRING;
END_ENTITY;
-- IfcProductExtension - Type Definitions
TYPE IfcBuildingTypeEnum = ENUMERATION OF (
    Building );

```

```

END_TYPE;
TYPE IfcBuildingStoreyTypeEnum = ENUMERATION OF (
    BuildingStorey );
END_TYPE;
TYPE IfcConnectionTypeEnum = ENUMERATION OF (
    AtPath
    ,Start
    ,AtEnd );
END_TYPE;
TYPE IfcContainmentTypeEnum = ENUMERATION OF (
    ProjectContainer
    ,SiteContainer
    ,BuildingContainer
    ,BuildingStoreyContainer
    ,SpaceContainer);
END_TYPE;
TYPE IfcElementFunctionTypeEnum = ENUMERATION OF (
    Supporting
    ,Enclosing
    ,Servicing
    ,Equipping );
END_TYPE;
TYPE IfcOpeningTypeEnum = ENUMERATION OF (
    Opening
    ,Passage
    ,Recess
    ,Chase );
END_TYPE;
TYPE IfcSiteTypeEnum = ENUMERATION OF (
    BuildingSite );
END_TYPE;
TYPE IfcSpaceTypeEnum = ENUMERATION OF (
    Occupied
    ,Technical
    ,Circulation );
END_TYPE;
TYPE IfcSystemTypeEnum = ENUMERATION OF (
    StructuralSystem
    ,EnclosingSystem
    ,DistributionSystem
    ,FurnishingSystem
    ,SpatialSystem );
END_TYPE;
TYPE IfcZoneTypeEnum = ENUMERATION OF (
    Thermal
    ,Daylighting
    ,Equipment);
END_TYPE;
-- IfcProductExtension - Entity Definitions
ENTITY IfcBuilding
SUBTYPE OF (IfcProduct);
    GenericType      : IfcBuildingTypeEnum;
    calcTotalHeight  : OPTIONAL IfcLengthMeasure;
    calcSiteCoverage : OPTIONAL IfcAreaMeasure;
    calcTotalVolume  : OPTIONAL IfcVolumeMeasure;
INVERSE
    Contains      : SET [0:?] OF IfcRelContains FOR RelatingObject;
    IsContainedBy  : SET [0:?] OF IfcRelContains FOR RelatedObjects;
    ServicedBySystems: SET [0:?] OF IfcRelServicesBuildings FOR RelatedObjects;
WHERE
    WR1: SIZEOF(QUERY(Temp <* IsContainedBy |
        Temp.RelationshipType = ProjectContainer)) = 1;
END_ENTITY;
ENTITY IfcBuildingElement
ABSTRACT SUPERTYPE OF (ONEOF
    (IfcBeam,IfcBuiltIn,IfcColumn,IfcCovering,IfcDiscreteElement,IfcDistributionElement,IfcDoor,IfcElectricalAppliance,IfcEquipment,IfcFixture,IfcFloor,IfcFurniture,IfcRoofSlab,IfcWall,IfcWindow))
SUBTYPE OF (IfcElement);

```

```

    HasMaterial      : OPTIONAL IfcMaterialSelect;
INVERSE
    ProvidesBoundaries : SET [0:?] OF IfcRelSeparatesSpaces FOR RelatingObject;
END_ENTITY;
ENTITY IfcBuildingStorey
    SUBTYPE OF (IfcProduct);
    GenericType      : IfcBuildingStoreyTypeEnum;
    Elevation        : IfcLengthMeasure;
    calcTotalHeight  : OPTIONAL IfcLengthMeasure;
    calcTotalArea    : OPTIONAL IfcAreaMeasure;
    calcTotalVolume  : OPTIONAL IfcVolumeMeasure;
INVERSE
    Contains         : SET [0:?] OF IfcRelContains FOR RelatingObject;
    IsContainedBy     : SET [0:?] OF IfcRelContains FOR RelatedObjects;
WHERE
    WR1: SIZEOF(QUERY(Temp <* IsContainedBy |
        Temp.RelationshipType = ProjectContainer)) = 1;
    WR2: SIZEOF(QUERY(Temp <* IsContainedBy |
        Temp.RelationshipType = BuildingContainer)) = 1;
END_ENTITY;
ENTITY IfcElement
    ABSTRACT SUPERTYPE OF (ONEOF(
        IfcOpeningElement
        ,IfcBuildingElement))
    SUBTYPE OF (IfcProduct);
    PerformedFunctions : SET [0:?] OF IfcElementFunctionTypeEnum;
INVERSE
    ConnectedTo       : SET [0:?] OF IfcRelConnectsElements FOR RelatingObject;
    ConnectedFrom     : SET [0:?] OF IfcRelConnectsElements FOR RelatedObject;
    HasOpenings       : SET [0:?] OF IfcRelVoidsElements FOR RelatingObject;
    FillsVoids        : SET [0:1] OF IfcRelFillsElements FOR RelatedObjects;
    IsAssemblyThrough : SET [0:1] OF IfcRelAssemblesElements FOR RelatingObject;
    PartOfAssembly    : SET [0:1] OF IfcRelAssemblesElements FOR RelatedObjects;
    IsContainedBy     : SET [0:?] OF IfcRelContains FOR RelatedObjects;
WHERE
    WR1: SIZEOF(QUERY(Temp <* IsContainedBy |
        Temp.RelationshipType = ProjectContainer)) = 1;
    WR2: SIZEOF(QUERY(Temp <* IsContainedBy |
        (((Temp.RelationshipType = SiteContainer)
        OR (Temp.RelationshipType = BuildingContainer)
        OR (Temp.RelationshipType = BuildingStoreyContainer)
        OR (Temp.RelationshipType = SpaceContainer))
        AND (Temp.ContainedOrReferenced = TRUE )))) = 1;
END_ENTITY;
ENTITY IfcOpeningElement
    SUBTYPE OF (IfcElement);
    GenericType      : IfcOpeningTypeEnum;
INVERSE
    VoidsElements : SET [0:1] OF IfcRelVoidsElements FOR RelatedObjects;
    HasFillings   : SET [0:?] OF IfcRelFillsElements FOR RelatingObject;
WHERE
    WR1: SIZEOF(QUERY( Temp <* SELF\IfcObject.TypeDefinitions |
        NOT(Temp.TypedClass = 'IfcOpeningElement')) = 0;
END_ENTITY;
ENTITY IfcRelAssemblesElements
    SUBTYPE OF (IfcRelationship1toN);
    -- SELF\IfcRelationship1toN.RelatingObject : IfcElement;
    -- SELF\IfcRelationship1toN.RelatedObjects : LIST [1:?] OF IfcElement;
END_ENTITY;
ENTITY IfcRelAssemblesSpaces
    SUBTYPE OF (IfcRelationship1toN);
    -- SELF\IfcRelationship1toN.RelatingObject : IfcSpace;
    -- SELF\IfcRelationship1toN.RelatedObjects : LIST [1:?] OF IfcSpace;
END_ENTITY;
ENTITY IfcRelConnectsElements
    SUPERTYPE OF (ONEOF(
        IfcRelConnectsPathElements))
    SUBTYPE OF (IfcRelationship1to1);

```

```

    ConnectionGeometry : IfcConnectionGeometry;
-- SELF\IfcRelationship1to1.RelatingObject : IfcElement;
-- SELF\IfcRelationship1to1.RelatedObject : IfcElement;
END_ENTITY;
ENTITY IfcRelConnectsPathElements
SUBTYPE OF (IfcRelConnectsElements);
    RelatingPriorities : LIST [0:RelatingLayerCount] OF INTEGER;
    RelatedPriorities : LIST [0:RelatedLayerCount] OF INTEGER;
    RelatingConnectionType : IfcConnectionTypeEnum;
    RelatedConnectionType : IfcConnectionTypeEnum;
DERIVE
    RelatingLayerCount : INTEGER
        := IfcNoOfLayers(RelatingObject);
    RelatedLayerCount : INTEGER
        := IfcNoOfLayers(RelatedObject);
END_ENTITY;
ENTITY IfcConnectionGeometry
ABSTRACT SUPERTYPE OF (ONEOF(
    IfcPointConnectionGeometry))
SUBTYPE OF (IfcControl);
END_ENTITY;
ENTITY IfcPointConnectionGeometry
SUBTYPE OF (IfcConnectionGeometry);
    PointOnRelatingElement : IfcCartesianPoint;
    PointOnRelatedElement : OPTIONAL IfcCartesianPoint;
END_ENTITY;
ENTITY IfcRelContains
SUBTYPE OF (IfcRelationship1toN);
    RelationshipType : IfcContainmentTypeEnum;
    ContainedOrReferenced : BOOLEAN;
WHERE
    WR1: ((RelationshipType = ProjectContainer) AND
        ('IFC150FINAL.IFCPROJECT' IN TYPEOF(SELF\IfcRelationship1toN.RelatingObject)))
        XOR (RelationshipType <> ProjectContainer);
    WR2: ((RelationshipType = SiteContainer) AND
        ('IFC150FINAL.IFCSITE' IN TYPEOF(SELF\IfcRelationship1toN.RelatingObject)) AND
        NOT('IFC150FINAL.IFCPROJECT' IN TYPEOF(SELF\IfcRelationship1toN.RelatedObjects)))
        XOR (RelationshipType <> SiteContainer);
    WR3: ((RelationshipType = BuildingContainer) AND
        ('IFC150FINAL.IFCBUILDING' IN TYPEOF(SELF\IfcRelationship1toN.RelatingObject)) AND
        NOT('IFC150FINAL.IFCPROJECT' IN TYPEOF(SELF\IfcRelationship1toN.RelatedObjects)) AND
        NOT('IFC150FINAL.IFCSITE' IN TYPEOF(SELF\IfcRelationship1toN.RelatedObjects)))
        XOR (RelationshipType <> BuildingContainer);
    WR4: ((RelationshipType = BuildingStoreyContainer) AND
        ('IFC150FINAL.IFCBUILDINGSTOREY' IN TYPEOF(SELF\IfcRelationship1toN.RelatingObject)) AND
        NOT('IFC150FINAL.IFCPROJECT' IN TYPEOF(SELF\IfcRelationship1toN.RelatedObjects)) AND
        NOT('IFC150FINAL.IFCSITE' IN TYPEOF(SELF\IfcRelationship1toN.RelatedObjects)) AND
        NOT('IFC150FINAL.IFCBUILDING' IN TYPEOF(SELF\IfcRelationship1toN.RelatedObjects)))
        XOR (RelationshipType <> BuildingStoreyContainer);
    WR5: ((RelationshipType = SpaceContainer) AND
        ('IFC150FINAL.IFCSPACE' IN TYPEOF(SELF\IfcRelationship1toN.RelatingObject)) AND
        NOT('IFC150FINAL.IFCPROJECT' IN TYPEOF(SELF\IfcRelationship1toN.RelatedObjects)) AND
        NOT('IFC150FINAL.IFCSITE' IN TYPEOF(SELF\IfcRelationship1toN.RelatedObjects)) AND
        NOT('IFC150FINAL.IFCBUILDING' IN TYPEOF(SELF\IfcRelationship1toN.RelatedObjects)) AND
        NOT('IFC150FINAL.IFCBUILDINGSTOREY' IN TYPEOF(SELF\IfcRelationship1toN.RelatedObjects)))
        XOR (RelationshipType <> SpaceContainer);
END_ENTITY;
ENTITY IfcRelFillsElements
SUBTYPE OF (IfcRelationship1toN);
-- SELF\IfcRelationship1toN.RelatingObject : IfcOpeningElement;
-- SELF\IfcRelationship1toN.RelatedObjects : LIST [1:?] OF IfcElement;
WHERE
    WR1 : SIZEOF (QUERY (temp <* SELF\IfcRelationship1toN.RelatedObjects |
        'IFC150FINAL.IFCOPENINGELEMENT' IN TYPEOF(temp))) = 0;
END_ENTITY;
ENTITY IfcRelSeparatesSpaces
SUBTYPE OF (IfcRelationship1toN);
-- SELF\IfcRelationship1toN.RelatingObject : IfcBuildingElement;

```

```

-- SELF\IfcRelationship1toN.RelatedObjects : LIST [1:?] OF IfcSpaceBoundary;
END_ENTITY;
ENTITY IfcRelServicesBuildings
  SUBTYPE OF (IfcRelationship1toN);
-- SELF\IfcRelationship1toN.RelatingObject : IfcSystem;
-- SELF\IfcRelationship1toN.RelatedObjects : LIST [1:?] OF IfcBuilding;
END_ENTITY;
ENTITY IfcRelVoidsElements
  SUBTYPE OF (IfcRelationship1toN);
-- SELF\IfcRelationship1toN.RelatingObject : IfcElement;
-- SELF\IfcRelationship1toN.RelatedObjects : LIST [1:?] OF IfcOpeningElement;
WHERE
  WR1 : NOT('IFC150FINAL.IFCOPENINGELEMENT' IN
    TYPEOF(SELF\IfcRelationship1toN.RelatingObject));
END_ENTITY;
ENTITY IfcSite
  SUBTYPE OF (IfcProduct);
  GenericType      : IfcSiteTypeEnum;
  RefLatitude      : IfcCompoundPlaneAngleMeasure;
  RefLongitude     : IfcCompoundPlaneAngleMeasure;
  RefElevation     : IfcLengthMeasure;
  calcSitePerimeter : OPTIONAL IfcPositiveLengthMeasure;
  calcSiteArea     : OPTIONAL IfcAreaMeasure;
INVERSE
  Contains      : SET [0:?] OF IfcRelContains FOR RelatingObject;
  IsContainedBy  : SET [0:?] OF IfcRelContains FOR RelatedObjects;
WHERE
  WR1 : SIZEOF(QUERY(Temp <* IsContainedBy |
    Temp.RelationshipType = ProjectContainer)) = 1;
END_ENTITY;
ENTITY IfcSpace
  SUBTYPE OF (IfcSpatialElement);
  GenericType      : IfcSpaceTypeEnum;
  BoundedBy        : LIST [1:?] OF IfcSpaceBoundary;
  calcTotalPerimeter : OPTIONAL IfcPositiveLengthMeasure;
  calcTotalArea     : OPTIONAL IfcAreaMeasure;
  calcTotalVolume   : OPTIONAL IfcVolumeMeasure;
  calcAverageHeight : OPTIONAL IfcPositiveLengthMeasure;
INVERSE
  IsAssemblyThrough : SET [0:1] OF IfcRelAssemblesSpaces FOR RelatingObject;
  PartOfAssembly    : SET [0:1] OF IfcRelAssemblesSpaces FOR RelatedObjects;
  Contains          : SET [0:?] OF IfcRelContains FOR RelatingObject;
  IsContainedBy     : SET [0:?] OF IfcRelContains FOR RelatedObjects;
WHERE
  WR1 : SIZEOF(QUERY( temp <* SELF\IfcObject.TypeDefinitions |
    NOT(temp.TypedClass = 'IfcSpace')) = 0;
  WR2 : SIZEOF(QUERY(Temp <* IsContainedBy |
    Temp.RelationshipType = ProjectContainer)) = 1;
  WR3 : SIZEOF(QUERY(Temp <* IsContainedBy |
    (((Temp.RelationshipType = SiteContainer)
    OR (Temp.RelationshipType = BuildingStoreyContainer))
    AND (Temp.ContainedOrReferenced = TRUE )))) = 1;
END_ENTITY;
ENTITY IfcSpaceBoundary
  SUBTYPE OF (IfcSpatialElement);
  PhysicalOrVirtual : BOOLEAN;
INVERSE
  Bounds      : SET[1:2] OF IfcSpace FOR BoundedBy;
  ProvidedBy   : SET[0:1] OF IfcRelSeparatesSpaces
    FOR RelatedObjects;
WHERE
  WR1 : ((PhysicalOrVirtual = TRUE) AND (HIINDEX(ProvidedBy) = 1))
    OR
    ((PhysicalOrVirtual = FALSE) AND (HIINDEX(ProvidedBy) = 0));
END_ENTITY;
ENTITY IfcSpatialElement
  ABSTRACT SUPERTYPE OF (ONEOF(
    IfcSpace

```



```

    ,IfcSpaceBoundary))
SUBTYPE OF (IfcProduct);
END_ENTITY;
ENTITY IfcSystem
SUBTYPE OF (IfcGroup);
    GenericType : IfcSystemTypeEnum;
INVERSE
    ServicesBuildings : IfcRelServicesBuildings FOR RelatingObject;
WHERE
    WR1: SIZEOF(QUERY( temp <* SELF\IfcObject.TypeDefinitions |
        NOT(temp.TypedClass = 'IfcSystem')))) = 0;
END_ENTITY;
ENTITY IfcZone
SUBTYPE OF (IfcGroup);
    GenericType : IfcZoneTypeEnum;
WHERE
    WR1: SIZEOF(QUERY( temp <* SELF\IfcGroup.GroupedBy.RelatedObjects |
        ('IFC150FINAL.IFCZONE' IN TYPEOF(temp)) OR
        ('IFC150FINAL.IFCSPACE' IN TYPEOF(temp))
    )) = HIINDEX(SELF\IfcGroup.GroupedBy.RelatedObjects);
    WR2: SIZEOF(QUERY( temp <* SELF\IfcObject.TypeDefinitions |
        NOT(temp.TypedClass = 'IfcZone')))) = 0;
END_ENTITY;
-- IfcProductExtension - Function Definitions
FUNCTION IfcNoOfLayers (
    Element : IfcElement )
    : INTEGER;
IF NOT(EXISTS(Element.HasMaterial)) THEN
    RETURN (?);
END_IF;
IF 'IFC150FINAL.IFCMATERIAL' IN TYPEOF(Element.HasMaterial) THEN
    RETURN (1);
END_IF;
IF 'IFC150FINAL.IFCMATERIALLAYERSET' IN TYPEOF(Element.HasMaterial) THEN
    RETURN (HIINDEX(Element.HasMaterial.HasMaterialLayers));
END_IF;
RETURN (?);
END_FUNCTION;
-- IfcPropertyResource - Type Specification for Actor
TYPE IfcRoleTypeEnum = ENUMERATION OF (
    Supplier
    ,Manufacturer
    ,Contractor
    ,SubContractor
    ,Architect
    ,StructuralEngineer
    ,ServicesEngineer
    ,CostEngineer
    ,Client
    ,BuildingOwner
    ,BuildingOperator
    ,Other);
END_TYPE;
TYPE IfcActorSelect = SELECT (
    IfcOrganization
    ,IfcPerson
    ,IfcPersonAndOrganization);
END_TYPE;
-- IfcPropertyResource - Type Specification for Material
TYPE IfcMaterialSelect = SELECT (
    IfcMaterialLayerSet
    ,IfcMaterialList);
END_TYPE;
-- IfcPropertyResource - Type Specification for Cost
TYPE IfcCostOperatorEnum = ENUMERATION OF (
    AddValue
    ,SubtractValue
    ,MultiplyValue

```

```

,AddPercent
,SubtractPercent
,MultiplyPercent);
END_TYPE;
TYPE IfcCostTypeEnum = ENUMERATION OF (
    LaborCost
    ,PlantCost
    ,MaterialCost
    ,SubContractCost
    ,PreliminariesCost
    ,PrimeCost
    ,BillOfMaterialsCost
    ,ProvisionalCost );
END_TYPE;
TYPE IfcCurrencyTypeEnum = ENUMERATION OF (
    AED, AES, ATS, AUD, BBD, BEG, BGL, BHD, BMD,
    BND, BRL, BSD, BWP, BZD, CAD, CBD, CHF, CLP,
    CNY, CYS, CZK, DDP, DEM, DKK, EGL, EST, FAK,
    FIM, FJD, FKP, FRF, GBP, GIP, GMD, GRX, HKD,
    HUF, ICK, IDR, ILS, INR, IRP, ITL, JMD, JOD,
    JPY, KES, KRW, KWD, KYD, LKR, LUF, MTL, MUR,
    MXN, MYR, NLG, NZD, OMR, PGK, PHP, PKR, PLN,
    PTN, QAR, RUR, SAR, SCR, SEK, SGD, SKP, THB,
    TRL, TTD, TWD, USD, VEB, VND, XEU, ZAR, ZWD);
END_TYPE;
TYPE IfcModifierBasisEnum = ENUMERATION OF (
    Running
    ,Static);
END_TYPE;
-- IfcPropertyResource - Type Specification for DateTime
TYPE IfcDayInMonthNumber = INTEGER;
END_TYPE;
TYPE IfcDaylightSavingNumber = INTEGER;
WHERE
    WR1: { 0 <= SELF <= 2 };
END_TYPE;
TYPE IfcHourInDay = INTEGER;
WHERE
    WR1: { 0 <= SELF < 24 };
END_TYPE;
TYPE IfcMinuteInHour = INTEGER;
WHERE
    WR1: { 0 <= SELF <= 59 };
END_TYPE;
TYPE IfcMonthInYearNumber = INTEGER;
WHERE
    WR1: { 1 <= SELF <= 12 };
END_TYPE;
TYPE IfcSecondInMinute = REAL;
WHERE
    WR1: { 0 <= SELF < 60 };
END_TYPE;
TYPE IfcYearNumber = INTEGER;
END_TYPE;
TYPE IfcDateTimeSelect = SELECT (
    IfcCalendarDate
    ,IfcLocalTime
    ,IfcDateAndTime );
END_TYPE;
-- IfcTypeDefResource - Entity Definitions for Actor
ENTITY IfcActorRole;
    Name      : IfcRoleTypeEnum;
    Description : OPTIONAL STRING;
END_ENTITY;
ENTITY IfcAddress;
    InternalLocation : OPTIONAL STRING;
    AddressLines    : LIST [0:?] OF STRING;
    Town            : OPTIONAL STRING;

```

```

    Region      : OPTIONAL STRING;
    PostalCode   : OPTIONAL STRING;
    Country      : OPTIONAL STRING;
    FacsimileNumbers : LIST [0:?] OF STRING;
    TelephoneNumbers : LIST [0:?] OF STRING;
    ElectronicMailAddresses : LIST [0:?] OF STRING;
    TelexNumber   : OPTIONAL STRING;
    WWWHomePage   : OPTIONAL STRING;
    Description    : OPTIONAL STRING;
  INVERSE
    OfPerson      : SET[0:?] OF IfcPerson FOR Addresses;
    OfOrganization : SET[0:?] OF IfcOrganization FOR Addresses;
  WHERE
    WR1: EXISTS (InternalLocation) OR
        (HIINDEX(AddressLines) > 0) OR
        EXISTS (Town) OR
        EXISTS (Region) OR
        EXISTS (PostalCode) OR
        EXISTS (Country) OR
        (HIINDEX(FacsimileNumbers) > 0) OR
        (HIINDEX(TelephoneNumbers) > 0) OR
        (HIINDEX(ElectronicMailAddresses) > 0) OR
        EXISTS (WWWHomePage);
  END_ENTITY;
  ENTITY IfcOrganization
  SUBTYPE OF (IfcProperty);
    Name      : STRING;
    Addresses  : LIST [0:?] OF IfcAddress;
    Roles      : LIST [0:?] OF IfcActorRole;
    Description : OPTIONAL STRING;
  END_ENTITY;
  ENTITY IfcPerson
  SUBTYPE OF (IfcProperty);
    FamilyName : OPTIONAL STRING;
    GivenName  : OPTIONAL STRING;
    MiddleNames : OPTIONAL STRING;
    PrefixTitles : OPTIONAL STRING;
    SuffixTitles : OPTIONAL STRING;
    Addresses   : LIST [0:?] OF IfcAddress;
    Roles        : LIST [0:?] OF IfcActorRole;
  WHERE
    WR1: EXISTS(FamilyName) OR EXISTS(GivenName);
  END_ENTITY;
  ENTITY IfcPersonAndOrganization
  SUBTYPE OF (IfcProperty);
    ThePerson      : IfcPerson;
    TheOrganization : IfcOrganization;
    Roles           : LIST [0:?] OF IfcActorRole;
  END_ENTITY;
  -- IfcPropertyResource - ENTITY Definitions for Material
  ENTITY IfcMaterial
  SUBTYPE OF (IfcProperty);
    MaterialName      : STRING;
    MaterialClassification : IfcClassificationList;
  INVERSE
    RegisteredBy      : IfcProjectMaterialRegistry FOR RegisteredMaterials;
  END_ENTITY;
  ENTITY IfcMaterialList
  SUBTYPE OF (IfcProperty);
    Materials          : IfcMaterial;
  END_ENTITY;
  ENTITY IfcMaterialLayer;
    Material           : IfcMaterial;
    OffsetFromMLSBase  : IfcPositiveLengthMeasure;
    LayerThickness     : IfcPositiveLengthMeasure;
    DefaultPriority     : INTEGER;
  END_ENTITY;
  ENTITY IfcMaterialLayerSet

```

```

SUBTYPE OF (IfcProperty);
  LayerSetName      : STRING;
  HasMaterialLayers : LIST [1:?] OF IfcMaterialLayer;
END_ENTITY;
ENTITY IfcMaterialLayerSetUsage;
  ForLayerSet      : IfcMaterialLayerSet;
  MlsOffsetFromBaseline: IfcLengthMeasure;
  MlsSenseLtoR     : BOOLEAN;
DERIVE
  TotalThickness      : IfcLengthMeasure
    := IfcMlsTotalThickness(ForLayerSet);
END_ENTITY;
ENTITY IfcProjectMaterialRegistry;
  RegisteredMaterials : LIST [0:?] OF IfcMaterial;
END_ENTITY;
-- IfcPropertyResource - Entity Definitions for Classification
ENTITY IfcClassification
SUBTYPE OF (IfcProperty);
  Source      : STRING;
  Table       : OPTIONAL STRING;
  Notation    : IfcClassificationNotation;
  Description  : STRING;
  Edition     : OPTIONAL STRING;
END_ENTITY;
ENTITY IfcClassificationList
SUBTYPE OF (IfcProperty);
  Classifications : LIST [1:?] OF IfcClassification;
  Priority        : INTEGER;
END_ENTITY;
ENTITY IfcClassificationNotation;
  NotationStrings: LIST [1:?] OF IfcNotationString;
END_ENTITY;
ENTITY IfcNotationString;
  NotationValue : STRING;
  Separator     : OPTIONAL STRING;
  Purpose       : OPTIONAL STRING;
END_ENTITY;
-- IfcPropertyResource - Entity Definitions for Cost
ENTITY IfcCost
SUBTYPE OF (IfcProperty);
  CostType      : IfcCostTypeEnum;
  BaseCostValue : OPTIONAL REAL;
  FinalCostValue : OPTIONAL REAL;
  Currency      : IfcCurrencyTypeEnum;
  ModifierBasis : OPTIONAL IfcModifierBasisEnum;
  CostModifiers : LIST[0:?] OF IfcCostModifier;
  UnitCostBasis : IfcMeasureWithUnit;
  CostDate      : OPTIONAL IfcDateTimeSelect;
  CostComponents : LIST [0:?] OF IfcCost;
END_ENTITY;
ENTITY IfcCostModifier;
  Purpose      : STRING;
  CostValue    : REAL;
  CostOperator : IfcCostOperatorEnum;
END_ENTITY;
-- IfcPropertyResource - Entity Definition for Date and Time
ENTITY IfcCalendarDate
SUBTYPE OF (IfcProperty);
  DayComponent  : IfcDayInMonthNumber;
  MonthComponent : IfcMonthInYearNumber;
  YearComponent : IfcYearNumber;
WHERE
  WR1: IfcValidCalendarDate (SELF);
END_ENTITY;
ENTITY IfcCoordinatedUniversalTimeOffset;
  HourOffset    : IfcHourInDay;
  MinuteOffset  : OPTIONAL IfcMinuteInHour;
  Ahead         : BOOLEAN;

```

```

END_ENTITY;
ENTITY IfcDateAndTime
SUBTYPE OF (IfcProperty);
  DateComponent : IfcCalendarDate;
  TimeComponent : IfcLocalTime;
END_ENTITY;
ENTITY IfcLocalTime
SUBTYPE OF (IfcProperty);
  HourComponent : IfcHourInDay;
  MinuteComponent : OPTIONAL IfcMinuteInHour;
  SecondComponent : OPTIONAL IfcSecondInMinute;
  Zone : IfcCoordinatedUniversalTimeOffset;
  DaylightSavingOffset : OPTIONAL IfcDaylightSavingNumber;
WHERE
  WR1: IfcValidTime (SELF);
END_ENTITY;
-- IfcPropertyResource - Function Definition for Material
FUNCTION IfcMlsTotalThickness (
  LayerSet : IfcMaterialLayerSet)
  : IfcLengthMeasure;
LOCAL
  Max : IfcLengthMeasure
  := LayerSet.HasMaterialLayers[1].OffsetFromMLSBase
  + LayerSet.HasMaterialLayers[1].LayerThickness;
  Min : IfcLengthMeasure
  := LayerSet.HasMaterialLayers[1].OffsetFromMLSBase;
END_LOCAL;
IF SIZEOF(LayerSet.HasMaterialLayers) > 1 THEN
  REPEAT i := 2 TO HIIndex(LayerSet.HasMaterialLayers);
    IF (LayerSet.HasMaterialLayers[i].OffsetFromMLSBase
      + LayerSet.HasMaterialLayers[i].LayerThickness) > Max
      THEN Max := LayerSet.HasMaterialLayers[i].OffsetFromMLSBase
      + LayerSet.HasMaterialLayers[i].LayerThickness;
    END_IF;
    IF LayerSet.HasMaterialLayers[i].OffsetFromMLSBase < Min
      THEN Min := LayerSet.HasMaterialLayers[i].OffsetFromMLSBase;
    END_IF;
  END_REPEAT;
END_IF;
RETURN (Max - Min);
END_FUNCTION;
-- IfcPropertyResource - Function Definition for Date and Time
FUNCTION IfcLeapYear (
  Year : IfcYearNumber)
  : BOOLEAN;
IF (((Year MOD 4) = 0) AND ((Year MOD 100) <> 0)) OR
  ((Year MOD 400) = 0) THEN
  RETURN(TRUE);
ELSE
  RETURN(FALSE);
END_IF;
END_FUNCTION;
FUNCTION IfcValidCalendarDate (
  Date : IfcCalendarDate)
  : LOGICAL;
IF NOT ({1 <= Date.DayComponent <= 31}) THEN
  RETURN(FALSE);
END_IF;
CASE Date.MonthComponent OF
  4 : RETURN({ 1<= Date.DayComponent <= 30});
  6 : RETURN({ 1<= Date.DayComponent <= 30});
  9 : RETURN({ 1<= Date.DayComponent <= 30});
  11 : RETURN({ 1<= Date.DayComponent <= 30});
  2 :
  BEGIN
    IF (IfcLeapYear(Date.YearComponent)) THEN
      RETURN({ 1<= Date.DayComponent <= 29});
    ELSE

```

```

    RETURN({ 1<= Date.DayComponent <= 28});
  END_IF;
END;
OTHERWISE : RETURN(TRUE);
END_CASE;
END_FUNCTION;
FUNCTION IfcValidTime (Time: IfcLocalTime) : BOOLEAN;
  IF EXISTS (Time.SecondComponent) THEN
    RETURN (EXISTS (Time.MinuteComponent));
  ELSE
    RETURN (TRUE);
  END_IF;
END_FUNCTION;
-- Entity Definitions for Type Def and Property Set
TYPE IfcTypeDefDomainViewEnum = ENUMERATION OF (
  ,CrossDomain
  ,Architecture
  ,HVAC
  ,FacMgmt );
END_TYPE;
-- Entity Definitions for Type Def and Property Set
ENTITY IfcObjectReference
  SUBTYPE OF (IfcProperty);
  Descriptor : STRING;
  ObjectReference : IfcProjectUniqueId;
END_ENTITY;
ENTITY IfcOccurrencePropertySet
  SUBTYPE OF (IfcPropertySet);
  TypeReference : IfcPropertyTypeDef;
WHERE
  WR1: SIZEOF(QUERY(temp <* SELF\IfcPropertySet.HasProperties |
    'IFC150FINAL.IFCPRODUCTSHAPE' IN TYPEOF(temp))) = 0;
END_ENTITY;
ENTITY IfcProperty
  ABSTRACT SUPERTYPE OF (ONEOF
    (IfcPropertySet,IfcObjectReference,IfcSimpleProperty,IfcProductShape,IfcCalendarDate,IfcClassification,IfcClassificationList,IfcCost,IfcDateAndTime,IfcLocalTime,IfcMaterial,IfcMaterialLayerSet,IfcMaterialList,IfcOrganization,IfcPerson,IfcPersonAndOrganization));
  INVERSE
    PartOfPropertySet : SET [0:1] OF IfcPropertySet
      FOR HasProperties;
END_ENTITY;
ENTITY IfcPropertySet
  SUPERTYPE OF (ONEOF(
    ,IfcSharedPropertySet
    ,IfcOccurrencePropertySet))
  SUBTYPE OF (IfcProperty);
  ProjectId : IfcProjectUniqueId;
  Descriptor : STRING;
  HasProperties : LIST [1:?] OF IfcProperty;
WHERE
  WR1: SIZEOF(QUERY(Temp <* HasProperties | Temp :=: SELF)) = 0;
END_ENTITY;
ENTITY IfcPropertyTypeDef;
  ProjectId : IfcProjectUniqueId;
  OwnerHistory : IfcOwnerHistory;
  TypeDefName : STRING;
  TypedClass : STRING;
  GenericType : OPTIONAL STRING;
  SpecificType : OPTIONAL STRING;
  TypeDomainView : IfcTypeDefDomainViewEnum;
  SharedProperties : LIST [0:?] OF IfcSharedPropertySet;
  GenericTypeReference : OPTIONAL IfcPropertyTypeDef;
  INVERSE
    ReferencedBySpecificType : SET [0:?] OF IfcPropertyTypeDef
      FOR GenericTypeReference;
    OccurrenceSets : SET [0:?] OF IfcOccurrencePropertySet
      FOR TypeReference;
WHERE

```

```

    WR1: EXISTS(GenericType) OR EXISTS(SpecificType);
END_ENTITY;
ENTITY IfcPropertyWithUnit
  SUBTYPE OF (IfcSimpleProperty);
  UnitComponent : IfcUnit;
END_ENTITY;
ENTITY IfcSharedPropertySet
  SUBTYPE OF (IfcPropertySet);
  INVERSE
    TypeDefined : IfcPropertyTypeDef
      FOR SharedProperties;
WHERE
  WR1: SIZEOF(QUERY(temp <* SELF\IfcPropertySet.HasProperties |
    'IFC150FINAL.IFCPRODUCTSHAPE' IN TYPEOF(temp))) = 0;
END_ENTITY;
ENTITY IfcSimpleProperty
  SUPERTYPE OF (ONEOF(IfcPropertyWithUnit))
  SUBTYPE OF (IfcProperty);
  Descriptor : STRING;
  ValueComponent : IfcMeasureValue;
END_ENTITY;
-- Type Definitions for Shape Representation
TYPE IfcRepDetailTypeEnum = ENUMERATION OF (
  Sketch
  ,Outline
  ,Design
  ,Detail
  ,Undefined);
END_TYPE;
TYPE IfcRepViewTypeEnum = ENUMERATION OF (
  Plan
  ,Section
  ,Elevation
  ,Isometric
  ,Diagrammatic
  ,Undefined);
END_TYPE;
TYPE IfcShapeRepTypeEnum = ENUMERATION OF (
  BoundingBox
  ,AttributeDriven
  ,Explicit);
END_TYPE;
TYPE IfcBooleanOperator = ENUMERATION OF (
  Union
  ,Difference
  ,Intersection);
END_TYPE;
TYPE IfcProductComponentShapeSelect = SELECT (
  IfcShapeBody
  ,IfcShapeResult);
END_TYPE;
-- Entity Definitions for Shape Representation
ENTITY IfcProductShape
  SUBTYPE OF (IfcProperty);
  ProjectId : IfcProjectUniquelId;
  ProductComponentShape : IfcProductComponentShapeSelect;
  Descriptor : OPTIONAL STRING;
END_ENTITY;
ENTITY IfcRepresentationContext;
  ProjectId : IfcProjectUniquelId;
  DetailType : IfcRepDetailTypeEnum;
  ViewType : IfcRepViewTypeEnum;
  Precision : INTEGER;
  INVERSE
    RepresentationsInContext : SET[1:?] OF IfcShapeRepresentation
      FOR ContextOfItems;
END_ENTITY;
ENTITY IfcShapeBody;

```

```

    ProjectId      : IfcProjectUniqueId;
    Representations : LIST [1:?] OF IfcShapeRepresentation;
    ComponentDescriptor: OPTIONAL STRING;
END_ENTITY;
ENTITY IfcShapeResult;
    ProjectId      : IfcProjectUniqueId;
    Operands       : LIST [2:?] OF IfcProductComponentShapeSelect;
    Operator       : IfcBooleanOperator;
    ComponentDescriptor: OPTIONAL STRING;
WHERE
    WR1: Operator <> IfcBooleanOperator.Intersection;
END_ENTITY;
ENTITY IfcShapeRepresentation;
    Items          : SET [1:?] OF IfcGeometricRepresentationItem;
    ContextOfItems : IfcRepresentationContext;
    RepresentationType : IfcShapeRepTypeEnum;
    UsageDescriptor : OPTIONAL STRING;
INVERSE
    OfShapeBody : IfcShapeBody FOR Representations;
WHERE
    WR1: ((RepresentationType = BoundingBox) AND (HIINDEX(Items) = 1) AND
        ('IFC150FINAL.IFCBOUNDINGBOX' IN TYPEOF(Items[1])))
        OR
        ((RepresentationType = AttributeDriven) AND (HIINDEX(Items) = 1) AND
        (('IFC150FINAL.IFCATTDIVENEXTRUDESOLID' IN TYPEOF(Items[1]))
        OR ('IFC150FINAL.IFCATTDIVENREVOLVEDSOLID' IN TYPEOF(Items[1]))))
        OR
        ((RepresentationType = Explicit) AND (SIZEOF(QUERY(temp <* Items |
        ('IFC150FINAL.IFCBOUNDINGBOX' IN TYPEOF(temp))
        OR ('IFC150FINAL.IFCATTDIVENEXTRUDESOLID' IN TYPEOF(temp))
        OR ('IFC150FINAL.IFCATTDIVENREVOLVEDSOLID' IN TYPEOF(temp)))) = 0));
END_ENTITY;
-- IfcSharedBldgElements - Type Definitions
TYPE IfcBeamTypeEnum = ENUMERATION OF (
    SimpleBeam
    ,CompoundBeam
    ,Truss);
END_TYPE;
TYPE IfcBuiltInTypeEnum = ENUMERATION OF (
    Cabinet
    ,CounterTop
    ,Railing);
END_TYPE;
TYPE IfcColumnTypeEnum = ENUMERATION OF (
    SimpleColumn
    ,CompoundColumn
    ,Truss);
END_TYPE;
TYPE IfcCoveringTypeEnum = ENUMERATION OF (
    Ceiling
    ,Flooring
    ,WallCovering);
END_TYPE;
TYPE IfcDoorTypeEnum = ENUMERATION OF (
    SingleSwing
    ,DoubleSwing
    ,Slide
    ,Rollup
    ,Revolving);
END_TYPE;
TYPE IfcFloorTypeEnum = ENUMERATION OF (
    SolidFloor
    ,LayeredFloor
    ,ElementedFloor);
END_TYPE;
TYPE IfcRoofSlabTypeEnum = ENUMERATION OF (
    SolidSlab
    ,LayeredSlab

```



```

,ElementedSlab);
END_TYPE;
TYPE IfcWallTypeEnum = ENUMERATION OF (
    SolidWall
    ,LayeredWall
    ,ElementedWall);
END_TYPE;
TYPE IfcWindowTypeEnum = ENUMERATION OF (
    FixedCasement
    ,Sliding
    ,Awning
    ,DoubleHung
    ,Casement
    ,Pivoting);
END_TYPE;
-- IfcSharedBldgElements - Entity Definitions
ENTITY IfcBeam
SUBTYPE OF (IfcBuildingElement);
    GenericType : IfcBeamTypeEnum;
WHERE
    WR1: SIZEOF(QUERY( Temp <* SELF\IfcObject.TypeDefinitions |
        NOT(Temp.TypedClass = 'IfcBeam')))) = 0;
    WR2: 'IFC150FINAL.IFCMATERIAL' IN TYPEOF(SELF\IfcBuildingElement.HasMaterial);
END_ENTITY;
ENTITY IfcBuiltIn
SUBTYPE OF (IfcBuildingElement);
    GenericType : IfcBuiltInTypeEnum;
WHERE
    WR1: SIZEOF(QUERY( Temp <* SELF\IfcObject.TypeDefinitions |
        NOT(Temp.TypedClass = 'IfcBuiltIn')))) = 0;
END_ENTITY;
ENTITY IfcColumn
SUBTYPE OF (IfcBuildingElement);
    GenericType : IfcColumnTypeEnum;
WHERE
    WR1: SIZEOF(QUERY( Temp <* SELF\IfcObject.TypeDefinitions |
        NOT(Temp.TypedClass = 'IfcColumn')))) = 0;
    WR2: 'IFC150FINAL.IFCMATERIAL' IN TYPEOF(SELF\IfcBuildingElement.HasMaterial);
END_ENTITY;
ENTITY IfcCovering
SUBTYPE OF (IfcBuildingElement);
    GenericType : IfcCoveringTypeEnum;
    LayerInformation : IfcMaterialLayerSetUsage;
DERIVE
    SELF\IfcBuildingElement.HasMaterial : IfcMaterialSelect
        := LayerInformation.ForLayerSet;
INVERSE
    Covers : IfcRelCoversBldgElements FOR RelatedObjects;
WHERE
    WR1: SIZEOF(QUERY( Temp <* SELF\IfcObject.TypeDefinitions |
        NOT(Temp.TypedClass = 'IfcCovering')))) = 0;
    WR2: 'IFC150FINAL.IFCMATERIALLAYERSET' IN
        TYPEOF(SELF\IfcBuildingElement.HasMaterial);
END_ENTITY;
ENTITY IfcDoor
SUBTYPE OF (IfcBuildingElement);
    GenericType : IfcDoorTypeEnum;
WHERE
    WR1: SIZEOF(QUERY( Temp <* SELF\IfcObject.TypeDefinitions |
        NOT(Temp.TypedClass = 'IfcDoor')))) = 0;
END_ENTITY;
ENTITY IfcFloor
SUBTYPE OF (IfcBuildingElement);
    GenericType : IfcFloorTypeEnum;
    LayerInformation : IfcMaterialLayerSetUsage;
DERIVE
    SELF\IfcBuildingElement.HasMaterial : IfcMaterialSelect
        := LayerInformation.ForLayerSet;

```

```

WHERE
  WR1: SIZEOF(QUERY( Temp <* SELF\IfcObject.TypeDefinitions |
    NOT(Temp.TypedClass = 'IfcFloor')))) = 0;
  WR2: 'IFC150FINAL.IFCMATERIALLAYERSET' IN
    TYPEOF(SELF\IfcBuildingElement.HasMaterial);
END_ENTITY;
ENTITY IfcRelCoversBldgElements
SUBTYPE OF (IfcRelationship1toN);
-- SELF\IfcRelationship1toN.RelatingObject : IfcBuildingElement;
-- SELF\IfcRelationship1toN.RelatedObjects : LIST [1:?] OF IfcCovering;
END_ENTITY;
ENTITY IfcRoofSlab
SUBTYPE OF (IfcBuildingElement);
  GenericType      : IfcRoofSlabTypeEnum;
  LayerInformation : IfcMaterialLayerSetUsage;
DERIVE
  SELF\IfcBuildingElement.HasMaterial : IfcMaterialSelect
    := LayerInformation.ForLayerSet;
WHERE
  WR1: SIZEOF(QUERY( Temp <* SELF\IfcObject.TypeDefinitions |
    NOT(Temp.TypedClass = 'IfcRoofSlab')))) = 0;
  WR2: 'IFC150FINAL.IFCMATERIALLAYERSET' IN
    TYPEOF(SELF\IfcBuildingElement.HasMaterial);
END_ENTITY;
ENTITY IfcWall
SUBTYPE OF (IfcBuildingElement);
  GenericType      : IfcWallTypeEnum;
  LayerInformation : IfcMaterialLayerSetUsage;
DERIVE
  SELF\IfcBuildingElement.HasMaterial : IfcMaterialSelect
    := LayerInformation.ForLayerSet;
WHERE
  WR1: SIZEOF(QUERY( Temp <* SELF\IfcObject.TypeDefinitions |
    NOT(Temp.TypedClass = 'IfcWall')))) = 0;
  WR2: 'IFC150FINAL.IFCMATERIALLAYERSET' IN
    TYPEOF(SELF\IfcBuildingElement.HasMaterial);
END_ENTITY;
ENTITY IfcWindow
SUBTYPE OF (IfcBuildingElement);
  GenericType      : IfcWindowTypeEnum;
WHERE
  WR1: SIZEOF(QUERY( Temp <* SELF\IfcObject.TypeDefinitions |
    NOT(Temp.TypedClass = 'IfcWindow')))) = 0;
END_ENTITY;
-- IfcSharedBldgElements - Type Definitions
TYPE IfcElectricalApplianceTypeEnum = ENUMERATION OF (
  ,Computer
  ,Copier
  ,Facsimile
  ,Printer
  ,Telephone);
END_TYPE;
TYPE IfcEquipmentTypeEnum = ENUMERATION OF (
  ,AirFilter
  ,AirHandler
  ,Boiler
  ,Chiller
  ,Coil
  ,Compressor
  ,Convactor
  ,CoolingTower
  ,Fan
  ,HeatExchanger
  ,Motor
  ,PackagedACUnit
  ,Pump
  ,TubeBundle
  ,UnitHeater);

```

```

END_TYPE;
TYPE IfcFixtureTypeEnum = ENUMERATION OF (
    Electrical
    ,Plumbing);
END_TYPE;
TYPE IfcDiscreteElementTypeEnum = ENUMERATION OF (
    Insulation);
END_TYPE;
TYPE IfcDistributionElementTypeEnum = ENUMERATION OF (
    NotYetDefined);
END_TYPE;
-- IfcSharedBldgServiceElements - Entity Definitions
ENTITY IfcElectricalAppliance
    SUBTYPE OF (IfcBuildingElement);
    GenericType : IfcElectricalApplianceTypeEnum;
END_ENTITY;
ENTITY IfcEquipment
    SUBTYPE OF (IfcBuildingElement);
    GenericType : IfcEquipmentTypeEnum;
END_ENTITY;
ENTITY IfcFixture
    SUBTYPE OF (IfcBuildingElement);
    GenericType : IfcFixtureTypeEnum;
END_ENTITY;
ENTITY IfcDiscreteElement
    SUBTYPE OF (IfcBuildingElement);
    GenericType : IfcDiscreteElementTypeEnum;
END_ENTITY;
ENTITY IfcDistributionElement
    SUBTYPE OF (IfcBuildingElement);
    GenericType : IfcDistributionElementTypeEnum;
END_ENTITY;
-- IfcUtilityResource - Type Definitions
TYPE IfcGloballyUniqueId = STRING;
END_TYPE;
TYPE IfcProjectUniqueId = STRING;
END_TYPE;
-- IfcUtilityResource - Entity Definitions
ENTITY IfcAuditTrail;
    CreationDate : IfcTimeStamp;
    DeletionDate : OPTIONAL IfcTimeStamp;
    CreatingUser : INTEGER;
    DeletingUser : OPTIONAL INTEGER;
    CreatingApplication : INTEGER;
    DeletingApplication : OPTIONAL INTEGER;
    AuditTrailLength : INTEGER;
    Transactions : LIST [0:AuditTrailLength] OF IfcTransaction;
INVERSE
    ToOwnerHistory : IfcOwnerHistory FOR AuditTrail;
WHERE
    WR1: AuditTrailLength <= 1;
END_ENTITY;
ENTITY IfcOwnerHistory;
    OwningActor : INTEGER;
    OwningApplication : INTEGER;
    ApplicationId : OPTIONAL STRING;
    OwnerDescriptor : OPTIONAL STRING;
    AuditTrail : OPTIONAL IfcAuditTrail;
END_ENTITY;
ENTITY IfcProjectAppRegistry;
    RegisteredApps : LIST [0:?] OF UNIQUE IfcRegisteredApplication;
END_ENTITY;
ENTITY IfcProjectTeamRegistry;
    RegisteredActors : LIST [0:?] OF UNIQUE IfcActorSelect;
END_ENTITY;
ENTITY IfcRegisteredApplication;
    ApplicationIdentifier: STRING(16);
    ApplicationFullName : STRING(255);

```

```

    ApplicationDeveloper : IfcActorSelect;
INVERSE
    RegisteredBy      : IfcProjectAppRegistry FOR RegisteredApps;
END_ENTITY;
ENTITY IfcTransaction;
    TransactionDate    : IfcTimeStamp;
    TransactingUser     : INTEGER;
    TransactingApplication : INTEGER;
INVERSE
    ToAuditTrail       : IfcAuditTrail FOR Transactions;
END_ENTITY;
ENTITY IfcTable;
    ProjectId          : IfcProjectUniqueId;
    Name                : STRING;
    Rows                : LIST [1:?] OF IfcTableRow;
DERIVE
    NumberOfCellsInRow : INTEGER
                        := HIINDEX(Rows[1].RowCells);
    NumberOfHeadings   : INTEGER
                        := SIZEOF(QUERY( Temp <* Rows | Temp.IsHeading));
    NumberOfDataRows   : INTEGER
                        := SIZEOF(QUERY( Temp <* Rows | NOT(Temp.IsHeading)));
WHERE
    WR2: SIZEOF(QUERY( Temp <* Rows |
        HIINDEX(Temp.RowCells) <> HIINDEX(Rows[1].RowCells)))
        = 0;
    WR3: { 0 <= NumberOfHeadings <= 1 };
END_ENTITY;
ENTITY IfcTableRow;
    RowCells           : LIST [1:?] OF IfcMeasureValue;
    IsHeading           : BOOLEAN;
INVERSE
    OfTable            : IfcTable FOR Rows;
END_ENTITY;
END_SCHEMA;

```

Appendix B: Modified Generated Files

The following is a listing of the automatically generated classes that were modified. These files can be found by searching for the string “MDS ADDED” through all Java files.

- Clfcaxis2placement.java
- Clfcaxis2placement2d.java
- Clfcaxis2placement3d.java
- ClfcboundingBox.java
- Clfcbuilding.java
- Clfcbuildingstorey.java
- Clfccartesianpoint.java
- Clfcdirection.java
- Clfcdoor.java
- Clfclocalplacement.java
- Clfcmeasurevalue.java
- Clfcproductshape.java
- Clfcproject.java
- Clfcpropertyset.java
- Clfcsharedpropertyset.java
- Clfcsimpleproperty.java
- Clfcsite.java
- Clfcspace.java
- Clfcwall.java
- Clfcwindow.java

Appendix C: Custom Class Code

The following are the custom classes that were created.

MDSAppInst

```
public interface Int_MDSAppInst extends SDAI.lang.App_inst {
    /** Returns string of guid (global unique identifier). */
    public String getGUID();
    /** Returns string of version ("versionMajor:versionMinor"). */
    public String getVersion();
    public long getVersionMajor();
    public long getVersionMinor();
    /** Returns string representation of the object. Created as:<p>
    * new String("[\" + getGUID() + \":Ver:\" + getVersion() + "]")
    */
    public String toString();
    /** Dumps object representation to System.out. */
    public void dump();
}

public abstract class MDSAppInst
    extends SDAI.COM.steptools.CApp_inst
    implements Int_MDSAppInst, Int_MDSDumpToIFC, Serializable {
    private long m_versionMajor;
    private long m_versionMinor;
    private boolean m_checkedOut;
    protected String m_guid;
    protected java.lang.String Projectid = null;
    // count used in creation of m_guid
    static transient private long count = 0;
    // boolean used during write to IFC file
    transient private boolean dumpingToIFC = false;

    /**
    * Default constructor. A m_guid is created during construction and m_versionMajor set to 1, m_versionMinor set to 0.
    */
    public MDSAppInst() {
        m_versionMajor = 1;
        m_versionMinor = 0;
        m_checkedOut = false;
        setGUID(generateGUID(classNameFinalPortion()));
        count += 1; // update counter
    }

    /**
    * Utility to return the final portion of a class name. If full class name is 'package1.package2.classA', this will return 'classA'.
    */
    public final String classNameFinalPortion() {
        // get final portion of class name as base for m_guid
        return getClass().getName().substring(getClass().getName().lastIndexOf('.') + 1);
    }

    /**
    * Routine to generate GUID. GUID is:
    * base + "_x_" + count + "_x_" + random number
    */
    private static String generateGUID(String base) {
        StringBuffer tmp = new StringBuffer(base);
        Random rand = new Random();
        tmp.append("_x_");
        tmp.append(count);
        tmp.append("_x_");
        tmp.append(rand.nextLong());
        return tmp.toString();
    }
}
```

```

/** Will remove this object from MDSObjectExtents if tracked. */
public boolean delete() {
    System.out.println("MDSAppInst:delete: " + this);
    if (MDSObjectExtents.getInstance() != null) {
        if (MDSObjectExtents.getInstance().findObject(this)) {
            MDSObjectExtents.getInstance().removeObject(this);
        }
    }
    return true;
}

public boolean checkOut() {
    m_checkedOut = true;
    return true;
}

public boolean checkIn() {
    m_versionMinor++;
    if (m_versionMinor == 100) {
        m_versionMajor++;
        m_versionMinor = 0;
    }
    m_checkedOut = true;
    return true;
}

private final void setGUID(String guid) {
    m_guid = guid;
    Projectid = guid;
}

public final String getGUID() { return m_guid;}
public final String getVersion() {
    return new String(String.valueOf(m_versionMajor) + "." + String.valueOf(m_versionMinor));
}

public final long getVersionMajor() { return m_versionMajor;}
public final long getVersionMinor() { return m_versionMinor;}
public void dump() {
    System.out.print("== OBJECT == " + classNameFinalPortion() + ", " + toString() + "\n");
}

public String toString() {
    return new String("[ " + getGUID() + " :Ver:" + getVersion() + "]" );
}

}

//=====
//=====
// routine to print to system.out the internal field values
public void dumpInternals() {
    processClassInternals(true);
}

// routine to retrieve the internal field values as a vector of strings
public Vector getInternals() {
    return processClassInternals(false);
}

private Vector processClassInternals(boolean display) {
    Vector internals = new Vector();
    try {
        Class c = getClass();
        if (display == true) System.out.println("Instance " + this);
        if (display == true) System.out.println("Class " + c);
        processFieldInternals(internals, display, c, this);
        Class superC = c.getSuperclass();
        while (superC != null) {
            if (display == true) System.out.println("Superclass " + superC);
            if (superC == Class.forName("java.lang.Object"))
                superC = null;
            else if (superC == Class.forName("SDAI.COM.steptools.CApp_inst"))
                superC = null;
            else {
                processFieldInternals(internals, display, superC, this);
                superC = superC.getSuperclass();
            }
        }
    }
    catch (Exception exc) {
        exc.printStackTrace();
    }
    return internals;
}

private void processFieldInternals(Vector internals, boolean display, Class c, Object inst) {
    String field;
    try {

```

```

        Field[] fields = c.getDeclaredFields();
        for (int i = 0; i < fields.length; i++) {
            try {
                Object obj = fields[i].get(this);
                field = new String(c.getName() + "." + fields[i].getName() + ": " + obj);
                internals.addElement(field);
                if (display == true)
                    System.out.println(c.getName() + "." + fields[i].getName() + ": " + obj);
                if (obj != null) {
                    Class cl = obj.getClass();
                    if (obj.getClass().isArray()) {
                        if (display == true)
                            System.out.println("\t" + obj.getClass().getComponentType().getName() +
                                "[" + Array.getLength(obj) + "]");
                    }
                }
            }
            catch (IllegalAccessException exc) {
                System.out.println(c + " IllegalAccessException, " + fields[i].getName());
            }
        }
    }
    catch (Exception exc) {
        exc.printStackTrace();
    }
}

//=====
//=====
// routine to dump to a part 21 file this instance and any attribute object variables
public void dumpToIFC(SDAI.lang.Model_contents mc) {
    if (dumpingToIFC) return;
    dumpingToIFC = true;
    try {
        // move this object to the model contents
        if (this instanceof SDAI.lang.App_inst) {
            if (classNameFinalPortion().startsWith("Clfc") ||
                classNameFinalPortion().startsWith("Elfc")) {
                mc.moveInstance((SDAI.lang.App_inst)this);
            }
        }
        // go up heirarchy chain adding parent class fields
        Class superC = getClass().getSuperclass();
        while (superC != null) {
            if (superC == Class.forName("java.lang.Object"))
                superC = null;
            else if (superC == Class.forName("SDAI.COM.steptools.CApp_inst"))
                superC = null;
            else if (superC == Class.forName("SDAI.COM.steptools.Slfc150final.MDSAppInst"))
                superC = null;
            else {
                dumpToIFCHelper(superC, this, mc);
                superC = superC.getSuperclass();
            }
        }
        // add main class fields
        dumpToIFCHelper(getClass(), this, mc);
        dumpingToIFC = false;
    }
    catch (Exception exc) {
        System.out.println("Exception Dumping - 1: " + this);
        exc.printStackTrace();
        dumpingToIFC = false;
    }
}

private void dumpToIFCHelper(Class c, Object inst, Model_contents mc) {
    try {
        // check for any fields to be moved to the model contents
        Field[] fields = c.getDeclaredFields();
        for (int i = 0; i < fields.length; i++) {
            try {
                Object obj = fields[i].get(inst);
                if (obj != null) {
                    if (obj instanceof Int_MDSDumpToIFC) {
                        ((Int_MDSDumpToIFC)obj).dumpToIFC(mc);
                    }
                    if (obj.getClass().isArray()) {
                        for (int j = 0; j < Array.getLength(obj); j++) {

```



```

        }
        System.out.println("\tDONE");
    }
    else
        System.out.println("\tType Definitions = NULL");
    // dump occurrence properties
    if (getOccurrenceproperties() != null) {
        System.out.println("\tOccurrence Properties:");
        Elfoccurrencepropertyset[] props = getOccurrenceproperties();
        for (int i = 0; i < props.length; i++) props[i].dump();
        System.out.println("\tDONE");
    }
    else
        System.out.println("\tOccurrence Properties = NULL");
    // dump extended properties
    if (getExtendedproperties() != null) {
        System.out.println("\tExtended Properties:");
        Elfpropertyset[] propSets = getExtendedproperties();
        for (int i = 0; i < propSets.length; i++) {
            //if (propSets[i].getDescriptor() != null) {
            System.out.println("\t\t" + propSets[i].getDescriptor() + ": " + propSets[i]);
            //}
            //else {
            //System.out.println("\t\t" + propSets[i]);
            //}
            if (propSets[i].getHasproperties() != null) {
                Elfproperty[] props = propSets[i].getHasproperties();
                for (int j = 0; j < props.length; j++) {
                    System.out.println("\t\t\t" + props[j]);
                }
            }
        }
        //
        for(int i = 0; i < props.length; i++) props[i].dump();
        System.out.println("\tDONE");
    }
    else
        System.out.println("\tExtended Properties = NULL");
}

//=====
// PROPERTY ROUTINES
//=====
// GENERAL PROPERTY ROUTINES
//=====
// PROPERTY TYPE DEFINITION OBJECT ROUTINES
//=====
/**
 * Returns an Elfpropertytypedef object with the given property type definition name.
 * Searches through the objects Elfpropertytypedef objects (gotten by calling getTypedefinitions()).
 * The return value is null if not found.
 */
public Elfpropertytypedef getPropertyTypeDef(String propertyTypeDefName) {
    if (propertyTypeDefName == null) {
        return null;
    }
    Elfpropertytypedef result = null;
    // look for given property type definition
    Elfpropertytypedef[] typeDefs = getTypedefinitions();
    if (typeDefs != null) {
        for (int i = 0; i < typeDefs.length; i++) {
            if (typeDefs[i].getTypedefname() != null) {
                if (typeDefs[i].getTypedefname().equals(propertyTypeDefName)) {
                    result = typeDefs[i];
                }
            }
        }
    }
    return result;
}

/**
 * Adds the given Elfpropertytypedef object.
 * Adds to the objects Elfpropertytypedef objects (gotten by calling getTypedefinitions()).
 * No check is first done to see if the property type definition already exists.
 */
public void addPropertyTypeDef(Elfpropertytypedef propertyTypeDef) {
    if (propertyTypeDef == null) {

```

```

        return;
    }
    int i;
    Elfpropertytypedef[] typeDefs = getTypedefinitions();
    Elfpropertytypedef[] typeDefsNew;
    if (typeDefs != null) {
        typeDefsNew = new Elfpropertytypedef[typeDefs.length + 1];
        for (i = 0; i < typeDefs.length; i++) typeDefsNew[i] = typeDefs[i];
        typeDefsNew[i] = propertyTypeDef;
    }
    else {
        typeDefsNew = new Elfpropertytypedef[] {propertyTypeDef};
    }
    setTypedefinitions(typeDefsNew);
}

/**
 * Removes the given Elfpropertytypedef object.
 * Searches through the objects Elfpropertytypedef objects (gotten by calling getTypedefinitions()).
 */
public void removePropertyTypeDef(Elfpropertytypedef propertyTypeDef) {
    if (propertyTypeDef == null) {
        return;
    }
    int i;
    Elfpropertytypedef[] typeDefs = getTypedefinitions();
    // look for propertyTypeDef
    int numFound = 0;
    if (typeDefs != null) {
        for (i = 0; i < typeDefs.length; i++) {
            if (typeDefs[i] == propertyTypeDef) numFound++;
        }
    }
    if (numFound == 0) {
        return;
    }
    // copy over non-matching type definitions
    int j = 0;
    Elfpropertytypedef[] typeDefsNew = new Elfpropertytypedef[typeDefs.length - numFound];
    for (i = 0; i < typeDefs.length; i++) {
        if (typeDefs[i] != propertyTypeDef) typeDefsNew[j++] = typeDefs[i];
    }
    setTypedefinitions(typeDefsNew);
}

//=====
// SHARED PROPERTY SET OBJECT ROUTINES
//=====
/**
 * Returns an Elfsharedpropertyset object with the given shared property set descriptor within the
 * given Elfpropertytypedef object.
 * Searches through the Elfpropertytypedef propertyTypeDef's Elfsharedpropertyset objects (gotten by calling getSharedproperties()).
 * The return value is null if not found.
 */
public Elfsharedpropertyset getSharedPropertySet(Elfpropertytypedef propertyTypeDef, String propertySetDescriptor) {
    if (propertyTypeDef == null) {
        return null;
    }
    Elfsharedpropertyset result = null;
    // look for given shared property set descriptor
    Elfsharedpropertyset[] sharedProps = propertyTypeDef.getSharedproperties();
    if (sharedProps != null) {
        for (int i = 0; i < sharedProps.length; i++) {
            if (sharedProps[i].getDescriptor() != null) {
                if (sharedProps[i].getDescriptor().equals(propertySetDescriptor)) {
                    result = sharedProps[i];
                }
            }
        }
    }
    return result;
}

/**
 * Returns an Elfsharedpropertyset object with the given shared property set descriptor within the
 * given Elfsharedpropertyset object.
 * Searches through the Elfsharedpropertyset sharedPropertySet's Elfproperty objects (gotten by calling getHasproperties()).
 * The return value is null if not found.
 */
public Elfsharedpropertyset getSharedPropertySet(Elfsharedpropertyset sharedPropertySet, String propertySetDescriptor) {

```

```

        if (sharedPropertySet == null) {
            return null;
        }
        ElfcsHaredPropertySet result = null;
        // look for given shared property set descriptor
        ElfCProperty[] props = sharedPropertySet.getHasProperties();
        if (props != null) {
            for (int i = 0; i < props.length; i++) {
                if (props[i] instanceof ElfcsHaredPropertySet) {
                    if (((ElfcsHaredPropertySet)props[i]).getDescriptor() != null) {
                        if (((ElfcsHaredPropertySet)props[i]).getDescriptor().equals(propertySetDescriptor)) {
                            result = (ElfcsHaredPropertySet)props[i];
                        }
                    }
                }
            }
        }
        return result;
    }
}

/**
 * Adds the given ElfcsHaredPropertySet object to the given ElfCPropertyTypeDef object.
 * Searches through the ElfCPropertyTypeDef propertyTypeDef's ElfcsHaredPropertySet objects (gotten by calling getSharedProperties()).
 * No check is first done to see if the shared property set already exists.
 */
public void addSharedPropertySet(ElfCPropertyTypeDef propertyTypeDef, ElfcsHaredPropertySet sharedPropertySet) {
    if (propertyTypeDef == null || sharedPropertySet == null) {
        return;
    }
    int i;
    ElfcsHaredPropertySet[] propSets = propertyTypeDef.getSharedProperties();
    ElfcsHaredPropertySet[] propSetsNew;
    if (propSets != null) {
        propSetsNew = new ElfcsHaredPropertySet[propSets.length + 1];
        for (i = 0; i < propSets.length; i++) propSetsNew[i] = propSets[i];
        propSetsNew[i] = sharedPropertySet;
    }
    else {
        propSetsNew = new ElfcsHaredPropertySet[] {sharedPropertySet};
    }
    propertyTypeDef.setSharedProperties(propSetsNew);
}

/**
 * Adds the given ElfcsHaredPropertySet object to the given ElfcsHaredPropertySet object.
 * Searches through the ElfcsHaredPropertySet sharedPropertySet's ElfCProperty objects (gotten by calling getHasProperties()).
 * No check is first done to see if the shared property set already exists.
 */
public void addSharedPropertySet(ElfcsHaredPropertySet sharedPropertySet1, ElfcsHaredPropertySet sharedPropertySet2) {
    if (sharedPropertySet1 == null || sharedPropertySet2 == null) {
        return;
    }
    int i;
    ElfCProperty[] props = sharedPropertySet1.getHasProperties();
    ElfCProperty[] propsNew;
    if (props != null) {
        propsNew = new ElfCProperty[props.length + 1];
        for (i = 0; i < props.length; i++) propsNew[i] = props[i];
        propsNew[i] = sharedPropertySet2;
    }
    else {
        propsNew = new ElfCProperty[] {sharedPropertySet2};
    }
    sharedPropertySet1.setHasProperties(propsNew);
}

/**
 * Removes the given ElfcsHaredPropertySet object from the given ElfCPropertyTypeDef object.
 * Searches through the ElfCPropertyTypeDef propertyTypeDef's ElfcsHaredPropertySet objects (gotten by calling getSharedProperties()).
 */
public void removeSharedPropertySet(ElfCPropertyTypeDef propertyTypeDef, ElfcsHaredPropertySet sharedPropertySet) {
    if (propertyTypeDef == null || sharedPropertySet == null) {
        return;
    }
    int i;
    ElfcsHaredPropertySet[] propSets = propertyTypeDef.getSharedProperties();
    // look for sharedPropertySet
    int numFound = 0;
    if (propSets != null) {
        for (i = 0; i < propSets.length; i++) {

```

```

        if (propSets[i] == sharedPropertySet) numFound++;
    }
}
if (numFound == 0) {
    return;
}
// copy over non-matching property sets
int j = 0;
ElfcsHaredPropertySet[] propSetsNew = new ElfcsHaredPropertySet[propSets.length - numFound];
for (i = 0; i < propSets.length; i++) {
    if (propSets[i] != sharedPropertySet) propSetsNew[j++] = propSets[i];
}
propertyTypeDef.setSharedProperties(propSetsNew);
}

/**
 * Removes the given ElfcsHaredPropertySet object from the given ElfcsHaredPropertySet object.
 * Searches through the ElfcsHaredPropertySet sharedPropertySet's ElfcsProperty objects (gotten by calling getHasProperties()).
 */
public void removeSharedPropertySet(ElfcsHaredPropertySet sharedPropertySet1, ElfcsHaredPropertySet sharedPropertySet2) {
    if (sharedPropertySet1 == null || sharedPropertySet2 == null) {
        return;
    }
    int i;
    ElfcsProperty[] props = sharedPropertySet1.getHasProperties();
    // look for sharedPropertySet2
    int numFound = 0;
    if (props != null) {
        for (i = 0; i < props.length; i++) {
            if (props[i] == sharedPropertySet2) numFound++;
        }
    }
    if (numFound == 0) {
        return;
    }
    // copy over non-matching property sets
    int j = 0;
    ElfcsProperty[] propsNew = new ElfcsProperty[props.length - numFound];
    for (i = 0; i < props.length; i++) {
        if (props[i] != sharedPropertySet2) propsNew[j++] = props[i];
    }
    sharedPropertySet1.setHasProperties(propsNew);
}

//=====
// EXTENDED PROPERTY OBJECT ROUTINES
//=====
/**
 * Returns an ElfcsPropertySet object with the given property set descriptor.
 * Searches through the objects ElfcsPropertySet objects (gotten by calling getExtendedProperties()).
 * The return value is null if not found.
 */
public ElfcsPropertySet getExtendedPropertySet(String propertySetDescriptor) {
    if (propertySetDescriptor == null) {
        return null;
    }
    ElfcsPropertySet result = null;
    if (getExtendedProperties() != null) {
        ElfcsPropertySet[] propSets = getExtendedProperties();
        for (int i = 0; i < propSets.length; i++) {
            if (propSets[i].getDescriptor().equals(propertySetDescriptor)) {
                result = propSets[i];
            }
        }
    }
    return result;
}

/**
 * Adds the given ElfcsPropertySet object.
 * Adds to the objects ElfcsPropertySet objects (gotten by calling getExtendedProperties()).
 * No check is first done to see if the extended property set already exists.
 */
public void addExtendedPropertySet(ElfcsPropertySet propertySet) {
    if (propertySet == null) {
        return;
    }
    ElfcsPropertySet[] propSets = getExtendedProperties();
    ElfcsPropertySet[] propSetsNew;
    if (propSets != null) {

```

```

        int i;
        propSetsNew = new Elfcpopertyset[propSets.length + 1];
        for (i = 0; i < propSets.length; i++) propSetsNew[i] = propSets[i];
        propSetsNew[i] = propertySet;
    }
    else {
        propSetsNew = new Elfcpopertyset[] {propertySet};
    }
    setExtendedproperties(propSetsNew);
}

/**
 * Removes the given Elfcpopertyset object.
 * Searches through the objects Elfcpopertyset objects (gotten by calling getExtendedproperties()).
 */
public void removeExtendedPropertySet(Elfcpopertyset propertySet) {
    if (propertySet == null) {
        return;
    }
    int i;
    Elfcpopertyset[] propSets = getExtendedproperties();
    // look for propertySet
    int numFound = 0;
    if (propSets != null) {
        for (i = 0; i < propSets.length; i++) {
            if (propSets[i] == propertySet) numFound++;
        }
    }
    if (numFound == 0) {
        return;
    }
    // copy over non-matching property sets
    int j = 0;
    Elfcpopertyset[] propSetsNew = new Elfcpopertyset[propSets.length - numFound];
    for (i = 0; i < propSets.length; i++) {
        if (propSets[i] != propertySet) propSetsNew[j++] = propSets[i];
    }
    setExtendedproperties(propSetsNew);
}

//=====
// PROPERTY SET OBJECT ROUTINES FOR ACCESSING SIMPLE PROPERTY OBJECTS
//=====
/**
 * Returns an Elfcsimpleproperty object with the given property set descriptor.
 * Searches through the Elfcpopertyset propertySet's Elfcpoperty objects (gotten by calling getHasproperties()).
 * The return value is null if not found.
 */
public Elfcsimpleproperty getSimpleProperty(Elfcpopertyset propertySet, String simplePropertyDescriptor) {
    if (propertySet == null || simplePropertyDescriptor == null) {
        return null;
    }
    Elfcsimpleproperty result = null;
    if (propertySet.getHasproperties() != null) {
        Elfcpoperty[] props = propertySet.getHasproperties();
        for (int i = 0; i < props.length; i++) {
            if (props[i] instanceof Elfcsimpleproperty) {
                if (((Elfcsimpleproperty)props[i]).getDescriptor().equals(simplePropertyDescriptor)) {
                    result = (Elfcsimpleproperty)props[i];
                }
            }
        }
    }
    return result;
}

/**
 * Adds the given Elfcsimpleproperty object to the given Elfcpopertyset object.
 * Adds to the Elfcpopertyset propertySet's Elfcpoperty objects (gotten by calling getHasproperties()).
 * No check is first done to see if the simple property already exists.
 */
public void addSimpleProperty(Elfcpopertyset propertySet, Elfcsimpleproperty property) {
    if (propertySet == null || property == null) {
        return;
    }
    Elfcpoperty[] props = propertySet.getHasproperties();
    Elfcpoperty[] propsNew;
    if (props != null) {
        int i;
        propsNew = new Elfcpoperty[props.length + 1];
    }

```

```

        for (i = 0; i < props.length; i++) propsNew[i] = props[i];
        propsNew[i] = property;
    }
    else {
        propsNew = new Elfproperty[] {property};
    }
    propertySet.setHasproperties(propsNew);
}

/**
 * Removes the given Elfcsimpleproperty object from the given Elfpropertyset object.
 * Searches through the Elfpropertyset propertySet's Elfproperty objects (gotten by calling getHasproperties()).
 */
public void removeSimpleProperty(Elfpropertyset propertySet, Elfcsimpleproperty property) {
    if (propertySet == null || property == null) {
        return;
    }
    int i;
    Elfproperty[] props = propertySet.getHasproperties();
    // look for property
    int numFound = 0;
    if (props != null) {
        for (i = 0; i < props.length; i++) {
            if (props[i] == property) numFound++;
        }
    }
    if (numFound == 0) {
        return;
    }
    // copy over non-matching properties
    int j = 0;
    Elfproperty[] propsNew = new Elfproperty[props.length - numFound];
    for (i = 0; i < props.length; i++) {
        if (props[i] != property) propsNew[j++] = props[i];
    }
    propertySet.setHasproperties(propsNew);
}

//=====
// DEFAULT MDS PROPERTY ROUTINES
//=====

private void setupMDSBasePropertySet() {
    // create property set
    Clfpropertyset propSet = new Clfpropertyset();
    propSet.setDescriptor(MDSObject.MDSBasePropertySet);
    addExtendedPropertySet(propSet);
}

private Elfpropertyset getMDSBasePropertySet() {
    return getExtendedPropertySet(MDSObject.MDSBasePropertySet);
}

private boolean isMDSBasePropertySet() {
    if (getMDSBasePropertySet() != null) {
        return true;
    }
    return false;
}

/**
 * Returns true if the given descriptor exists in a default shared property set; MDSObject.MDSBasePropertySet.
 */
public boolean existsProperty(String descriptor) {
    if (isMDSBasePropertySet() == false) return false;
    Elfpropertyset mdsBasePropSet = getMDSBasePropertySet();
    return getSimpleProperty(mdsBasePropSet, descriptor) != null;
}

/**
 * Removes the given descriptor from a default shared property set; MDSObject.MDSBasePropertySet.
 * This will remove all properties with the given descriptor.
 */
public void removeProperty(String descriptor) {
    if (isMDSBasePropertySet() == false) return;
    Elfpropertyset mdsBasePropSet = getMDSBasePropertySet();
    Elfcsimpleproperty prop = getSimpleProperty(mdsBasePropSet, descriptor);
    if (prop != null) {
        removeSimpleProperty(mdsBasePropSet, prop);
    }
}

/**
 * Adds a new property in a default shared property set; MDSObject.MDSBasePropertySet.
 * No check is made first if the descriptor already exists!

```

```

* Sets the value as an IFCDescriptiveMeasure.
*/
    public void addStringProperty(String descriptor, String value) {
        if (isMDSBasePropertySet() == false) setupMDSBasePropertySet();
        Elfcpropertyset mdsBasePropSet = getMDSBasePropertySet();
        Elfcsimpleproperty prop = getSimpleProperty(mdsBasePropSet, descriptor);
        if (prop == null) {
            prop = new Clfcsimpleproperty();
        }
        prop.setDescriptor(descriptor);
        prop.getValueComponent().setIfcdescriptivemeasure(value);
        addSimpleProperty(mdsBasePropSet, prop);
    }

/**
 * Adds a new property in a default shared property set; MDSObject.MDSBasePropertySet.
 * No check is made first if the descriptor already exists!
 * Sets the value as an IFCNumericMeasure.
 */
    public void addDoubleProperty(String descriptor, Double value) {
        if (isMDSBasePropertySet() == false) setupMDSBasePropertySet();
        Elfcpropertyset mdsBasePropSet = getMDSBasePropertySet();
        Elfcsimpleproperty prop = getSimpleProperty(mdsBasePropSet, descriptor);
        if (prop == null) {
            prop = new Clfcsimpleproperty();
        }
        prop.setDescriptor(descriptor);
        prop.getValueComponent().setIfcnumericmeasure(value.doubleValue());
        addSimpleProperty(mdsBasePropSet, prop);
    }

/**
 * Gets the property type of the property with this descriptor in a default shared property set; MDSObject.MDSBasePropertySet.
 * The string returned relates to data types defined in the IFC documentation for IFCMeasureValue. Ex: <p>
<pre>
"lflengthmeasure"
"lfcmassmeasure"
"lfcmeasure"
"lfcelectriccurrentmeasure"
"lfcthermodynamictemperaturemeasure"
"lfcamountofsubstancemeasure"
"lfcuminousintensitymeasure"
"lfcplaneanglemeasure"
"lfcsolidanglemeasure"
"lfcareameasure"
"lfcvolumemeasure"
"lfcrationmeasure"
"lfcparametervalue"
"lfcnumericmeasure"
"lfccontextdependentmeasure"
"lfcdescriptivemeasure"
"lfcpositivelengthmeasure"
"lfcpositiveplaneanglemeasure"
"lfcpositiveratiomeasure"
"lfccountmeasure"
"lfccompoundplaneanglemeasure"
"lftimedurationmeasure"
"lfctimestamp"
</pre>
 * The return value is null if property not found.
 */
    public String getPropertyType(String descriptor) {
        if (isMDSBasePropertySet() == false) return null;
        Elfcpropertyset mdsBasePropSet = getMDSBasePropertySet();
        Elfcsimpleproperty prop = getSimpleProperty(mdsBasePropSet, descriptor);
        if (prop != null) {
            return prop.getValueComponent().getUnderlying_type();
        }
        return null;
    }

/**
 * Gets a property in a default shared property set; MDSObject.MDSBasePropertySet.
 * Returns the first matching descriptor!
 * The return value is null if property not found.
 */
    public String getStringProperty(String descriptor) {
        if (isMDSBasePropertySet() == false) return null;
        Elfcpropertyset mdsBasePropSet = getMDSBasePropertySet();
        Elfcsimpleproperty prop = getSimpleProperty(mdsBasePropSet, descriptor);

```



```

        if (prop != null) {
            return prop.getValueComponent().getIfcdescriptivemeasure();
        }
        return null;
    }
}

/**
 * Gets a property in a default shared property set; MDSObject.MDSBasePropertySet.
 * Returns the first matching descriptor!
 * The return value is null if property not found.
 */
public Double getDoubleProperty(String descriptor) {
    if (isMDSBasePropertySet() == false) return null;
    Elfcpropertyset mdsBasePropSet = getMDSBasePropertySet();
    Elfcsimpleproperty prop = getSimpleProperty(mdsBasePropSet, descriptor);
    if (prop != null) {
        return new Double(prop.getValueComponent().getIfcnumericmeasure());
    }
    return null;
}

/**
 * Gets a vector of descriptors in a default shared property set; MDSObject.MDSBasePropertySet.
 */
public Vector getPropertyTitles() {
    Vector result = new Vector();
    if (isMDSBasePropertySet() == false) return result;
    Elfcpropertyset mdsBasePropSet = getMDSBasePropertySet();
    Elfcproperty[] props = mdsBasePropSet.getProperties();
    if (props != null) {
        for (int i = 0; i < props.length; i++) {
            result.addElement(((Elfcsimpleproperty)props[i]).getDescriptor());
        }
    }
    return result;
}
}

```

MDSProduct

```

public abstract class MDSProduct extends MDSObject implements Int_MDSProduct {
    /**
     * A local placement object is automatically created.
     * A bounding box product shape (size 0, 0, 0) is automatically created.
     */
    public MDSProduct() {
        setLocalplacement(new Clfclocalplacement(new Clfcaxis2placement3d(new Clfccartesianpoint(0, 0, 0)));
        setProductshape(Clfcproductshape.productshape_boundingBox_create(0, 0, 0));
    }

    //-----
    // these methods are redefined in subclasses!
    public Elfclocalplacement getLocalplacement() {
        System.out.println("MDSProduct:getLocalplacement, SHOULD NOT BE HERE!");
        return null;
    }
    public void setLocalplacement(Elfclocalplacement v) {
        System.out.println("MDSProduct:setLocalplacement, SHOULD NOT BE HERE!");
        return;
    }
    public Elfcproductshape getProductshape() {
        System.out.println("MDSProduct:getProductshape, SHOULD NOT BE HERE!");
        return null;
    }
    public void setProductshape(Elfcproductshape v) {
        System.out.println("MDSProduct:setProductshape, SHOULD NOT BE HERE!");
        return;
    }
}

//-----
//=====
// LOCATION ROUTINES
//=====
public void setRelativeLocation(Elfccartesianpoint location) {
    if (location == null) return;
    if (getLocalplacement() == null) return;
    ((Clfclocalplacement)getLocalplacement()).setRelativePlacementLocation(location);
}

```

```

    }
    public void setAbsoluteLocation(Elfcartesianpoint location) {
        if (location == null) return;
        if (getLocalplacement() == null) return;
        if (getLocalplacement().getPlacemntrelto() == null) return;
        if (getLocalplacement().getPlacemntrelto().isIfcproduct() == SDAI.lang.LOGICAL.TRUE)
            setRelativeLocation(Clfcartesianpoint.delete(location, getLocalplacement().getPlacemntrelto().getIfcproduct().getAbsoluteLocation()));
        setRelativeLocation(location);
    }
    public Elfcartesianpoint getRelativeLocation() {
        if (getLocalplacement() == null) return null;
        return((Clfclocalplacement)getLocalplacement()).getRelativePlacementLocation();
    }
    public Elfcartesianpoint getAbsoluteLocation() {
        if (getLocalplacement() == null) return null;
        if (getLocalplacement().getPlacemntrelto() == null) return null;
        if (getLocalplacement().getPlacemntrelto().isIfcproduct() == SDAI.lang.LOGICAL.TRUE)
            return Clfcartesianpoint.add(getLocalplacement().getPlacemntrelto().getIfcproduct().getAbsoluteLocation(), getRelativeLocation());
        return getRelativeLocation();
    }
}
//=====
// ANGLE ROUTINES
//=====
    public void setRelativeAngle(double angle) {
        if (getLocalplacement() == null) return;
        ((Clfclocalplacement)getLocalplacement()).setRelativePlacementAngle(angle);
    }
    public void setAbsoluteAngle(double angle) {
        if (getLocalplacement() == null) return;
        if (getLocalplacement().getPlacemntrelto() == null) return;
        if (getLocalplacement().getPlacemntrelto().isIfcproduct() == SDAI.lang.LOGICAL.TRUE)
            setRelativeAngle(angle - getLocalplacement().getPlacemntrelto().getIfcproduct().getAbsoluteAngle());
        setRelativeAngle(angle);
    }
    public double getRelativeAngle() {
        if (getLocalplacement() == null) return 0.0;
        return((Clfclocalplacement)getLocalplacement()).getRelativePlacementAngle();
    }
    public double getAbsoluteAngle() {
        if (getLocalplacement() == null) return 0.0;
        if (getLocalplacement().getPlacemntrelto() == null) return 0.0;
        if (getLocalplacement().getPlacemntrelto().isIfcproduct() == SDAI.lang.LOGICAL.TRUE)
            return(getLocalplacement().getPlacemntrelto().getIfcproduct().getAbsoluteAngle() + getRelativeAngle());
        return getRelativeAngle();
    }
    public void dump() {
        super.dump();
        if (getLocalplacement() != null) {
            System.out.print("\tLocalplacement:\n"); getLocalplacement().dump();
        }
        System.out.print("\tLocation:");
        if (getRelativeLocation() != null) {
            System.out.print(" Relative: " + getRelativeLocation());
        }
        if (getAbsoluteLocation() != null) {
            System.out.print(" Absolute: " + getAbsoluteLocation());
        }
        System.out.print("\n");
        System.out.print("\tAngle:");
        System.out.print(" Relative: " + Double.toString(getRelativeAngle()));
        System.out.print(" Absolute: " + Double.toString(getAbsoluteAngle()));
        System.out.print("\n");
        /*
        if (getProductshape() != null) {
            System.out.println("\tBounding Box: Xdim: " + Double.toString(getBoundingXDim()) +
                ", Ydim: " + Double.toString(getBoundingYDim()) +
                ", Zdim: " + Double.toString(getBoundingZDim()));
        }
        */
    }
}
}

```

Appendix D: PSE Post-Processing Batch File

This is the batch file that is used to post-process the object-model and it's helper classes. The order that these are done is important. It was necessary to increase the amount of memory that is used by the Java virtual machine (reason for –mx32m option).

```
java -mx32m com.odi.filter.OSCFP -verbose -dest . -inplace
    -nooptimizeclassinfo
    -pc SDAI.COM.steptools.CSelect SDAI.COM.steptools.CApp_inst
    -tf SDAI.COM.steptools.CApp_inst.owner
    -tf SDAI.COM.steptools.CApp_inst.persistentID
java -mx32m com.odi.filter.OSCFP -verbose -dest . -inplace
    MDSInterfaces*.class sdai\sifc150final*.class
    sdai\com\steptools\sifc150final\Cifc*.class
    sdai\com\steptools\sifc150final\Inv*.class
    sdai\com\steptools\sifc150final\MDS*.class
java -mx32m com.odi.filter.OSCFP -verbose -dest . -inplace
    -pa SDAI.COM.steptools.Sifc150final.Init
    SDAI.COM.steptools.Sifc150final.Roselnit
    SDAI.COM.steptools.Sifc150final.P21
java -mx32m com.odi.filter.OSCFP -verbose -dest . -inplace
    -pa MDSControllers*.class
```

Appendix E: Microsoft Access Tables

The following are the tables and their structure for the Microsoft Access database that is read from within the *Building Composer* core library.

Table: ActivityHeaders

Description: Activity Definitions. Table read from BuildingComposer!

Columns

Name	Type	Size
Name	Text	50
Description	ext	100

Table: BuildingHeaders

Description: Building Definitions. Table read from BuildingComposer!

Columns

Name	Type	Size
ID	Text	50
Description	Text	250

Table: FunctionHeaders

Description: Function Definitions.

Columns

Name	Type	Size
ID	Text	15
Name	Text	50
Description	Text	100
ActivityName	Text	50
Height	Double	8

Table: FunctionParentFunctionLinks

Description: Link a parent function with a child function.

Columns

Name	Type	Size
FunctionName	Text	50
ParentFunctionName	Text	50

Table: FunctionReqSetLinks

Description: Link a function with a requirement set. Table read from Building-Composer!

Columns

Name	Type	Size
FunctionName	Text	50
ReqSetName	Text	50

Table: FunctionRequirementLink

Columns

Name	Type	Size
FunctionName	Text	50
RequirementName	Text	50
Value	Text	255
ValueCanBeChanged	Yes/No	1
Source	Text	255

Table: FunctionSubFunctionLinks

Description: Link a super function with a sub function.

Columns

Name	Type	Size
FunctionName	Text	50
SubFunctionName	Text	50

Table: LibraryInfo

Properties

Description: Library Information. Table read from BuildingComposer!

Columns

Name	Type	Size
ID	Long Integer	4
Version	Text	50
Date	Date/Time	8
Name	Text	50
Description	Text	250
EnglishUnits	Yes/No	1
Client	Text	50
Comment	Text	250
UpdatedBy	Text	50

Table: RequirementHeaders

Description: Requirement Definitions. Table read from BuildingComposer!

Columns

Name	Type	Size
ID	Text	50
DisplayName	Text	50
Description	Text	200
RequirementSetName	Text	50
Level	Long Integer	4
Datatype	Long Integer	4
Value	Text	255
UOM	Text	50
IsRequired	Yes/No	1
ValueCanBeChanged	Yes/No	1
MultiValue	Yes/No	1
Source	Text	200

Table: RequirementMultiValues

Description: MultiValue Definitions. Table read from BuildingComposer!

Columns

Name	Type	Size
ID	Long Integer	4
RequirementName	Text	50
Value	Text	50

Table: RequirementSetHeadersDescription: Requirement Set Definitions. Table read from *Building Composer*!

Columns

Name	Type	Size
Name	Text	50
Description	Text	200
TabName	Text	15

Appendix F: Additional Information on Library Creation and Usage

Process for reading/processing/using application/extension data:

1. After the application is started, existing function templates (with their requirement sets and requirements) are read into the main default library for *Building Composer*. This is done so that all function templates reference the same requirement sets and requirements. Mainly this is so that many similar objects are not created in the database. It is desirable for all functions in the library to use the same requirement sets and requirements, whether existing or new.
2. The main Access database is read for *Building Composer*.
 - a. The activities are read in (MDSActivity objects created).
 - b. The requirement sets are read in (MDSRequirementSet objects are created). They are put into a generic requirement set vector and a function requirement set vector. The function requirement set vector is used when a function instance is created. The generic requirement set vector is used all other times. Requirement sets and requirements read in step 1 are not added. An example of why this is done follows. Suppose there is a requirement set with both function and non-function requirements. If that requirement set is told in step 2-d to be specifically linked to a function, then all of those requirements will be added. Therefore it makes sense to split function requirements from non-function requirements.
 - c. The requirements are read in (MDSProperty objects created). If they are generic, then they are linked to requirement sets in the generic vector. If they are for a function they are linked to requirement sets in the function vector.
 - d. The functions are read in (MDSFunctionTemplate objects created). Specifically linked functions/requirement sets (table FunctionReqSetLinks) are done.
 - e. Function/Property override values are read in.
 - f. The function templates are updated to have all required requirements. This is done so that when a function template is highlighted in the Import dialog, it shows the requirements that will be loaded.

3. When an extension is loaded, the data from its Access database is read into the main default library for *Building Composer* (following step 2).
4. When a non-function instance object is created, the main default library for *Building Composer* is searched for requirements to add (using requirement sets in the generic vector).
5. When a function instance object is created:
 - a. Requirement sets and requirements are copied from the function template.
 - b. The main default library for *Building Composer* is searched for requirements to add (using requirement sets in the function vector).
 - c. The function/property override values are checked.
 - d. When a combobox is required for requirement values, the library related to the extension is searched for multivalue options (table Requirement-MultiValues).

CERL Distribution

HQUSACE

ATTN: CECW-EE (2)

Chief of Engineers

ATTN: CEHEC-IM-LH (2)

Engineer Research and Development Center (Libraries)

ATTN: ERDC, Vicksburg, MS

ATTN: Cold Regions Research, Hanover, NH

ATTN: Topographic Engineering Center, Alexandria, VA

Defense Tech Info Center 22304

ATTN: DTIC-O

7

11/02

